Low-Cost Electrochemical Sensor System

MSU Advanced Microsystems/Controls Research Group

Dr. Andrew Mason – Sponsor Contact
Dr. Terrence Brown – Facilitator

Design Team 2:

Luke LaPointe – Web Design
Nick Timpf – Manager
Brent Woodman – Presentation Prep.
Steve Zuraski – Lab Coordinator
Executive Summary

Dr. Mason tasked design team two with creating a low cost alternative to current lab-based electro-chemical testing equipment. The product was required to be a portable battery powered device, which can easily be used in any location not just a laboratory.

The device allows users to customize and perform multiple electro-chemical tests. It operates using two AA batteries and communicates wirelessly with a laptop via a Bluetooth connection. This product has been designed to be easily customized for testing applications in fields such as: chemical, medical, and food safety.

The cost to construct this prototype was $120. This easily meets the required final cost in mass production of less than $100 set by the sponsor.

Acknowledgment

Design Team 2 would like to thank all those who helped in the creation of our final product. Our sponsor, Dr. Mason, created an open communication forum that allowed our team access to all of his resources including his lab and staff. Without this assistance we would not have been as successful. Specifically Xiaowen Liu, our technical liaison was invaluable. She found answers to every question we had throughout the semester as well as being our resident chemist. Dr. Brown our facilitator was absolutely great to work with. He provided the motivation to push the project forward even when we were not sure of the direction.

Technical assistance was provided by Brian Wright and Gregg Mulder. Both provided critical advice as well as numerous ideas to help finish our project. Their expertise was much appreciated.

Thanks to Dr. Wierzba for his crucial advice and willingness to answer questions regarding analog circuitry.

Additionally we would like to thank design teams one and six for making our experience in the lab a pleasant and light working environment. This made our semester not only tolerable but memorable.

Finally we would like to thank all of our friends and family for not renting out or repurposing our rooms over our four month absence.
# Table of Contents

- **Chapter 1**
  - Background Information p4
    - Electrochemical Testing
    - Potentiostat
    - Cyclic Voltammetry
    - Chronoamperometry
    - Low Frequency Electro-chemical Impedance Spectroscopy
  - Introduction p5

- **Chapter 2**
  - FAST Diagram p6
  - House of Quality p7
  - Solution Selection Matrix p8
  - dsPIC30F2020
  - DAC
  - Digital Potentiometer
  - Op-amps p9
  - DC/DC Converter/ Voltage Regulator
  - Bluetooth
  - LabVIEW
  - Budget p10

- **Chapter 3** p11
  - Potentiostat
  - Microprocessor p13
  - LabVIEW p16
  - EAGLE p18
  - Problems and Solutions P18
  - Power Layout p20

- **Chapter 4** p21
  - Data & Results

- **Chapter 5** p24
  - Summary
  - Schedule p25
  - Conclusion p26

- **Appendix I** p26
- **Appendix II** p32
- **Appendix III** p33
Chapter 1

Background Information:

Electrochemical testing:
Applying an external potential to an electrode either from a DC battery of constant potential or from a signal generator of varying potential to characterize a solution is electrochemical testing. These tests are normally used to characterize the corrosion of metal in a solution or determine the rate of oxidation of a solution (redox). This is directly related to the concentration of metal ions in a solution.

Potentiostat:
This is an electrical device used in electrochemical tests that keeps a constant voltage potential across a pair of electrodes. In a three electrode probe, the working electrode is held at constant potential with respect to the reference while the current between the working and counter electrode is measured.

Cyclic Voltammetry:
In this test a triangle stimulus is imposed on working electrode and the current is then measured. The current measurements are then plotted vs. voltage values to create a hysteresis curve. This test provides information regarding how fast the state of the solution can fluctuate in relation to the rate of change of the stimulus. The results of the cyclic voltammetry hysteresis curve can yield the proper settings on which to run the chronoamperometry test to characterize a solution.

Chronoamperometry:
Chronoamperometry is an electrochemical measuring technique used for the analysis of electrolyte solutions. A fast rising step in voltage is applied on the working electrode of an electrochemical cell and the current is then measured. The decay of the resulting current spike created between the counter and working electrodes is used to determine the concentration of ions in the solution. These decay time are compared to known values in a library of previous tests to classify the solution.

Low frequency Electrochemical Impedance Spectroscopy (EIS)
“Impedance spectroscopy can be used to investigate the various physical processes occurring at the electrode such as diffusion, charge transfer and double layer capacitance.” In systems such as cellular bi-lipid layer membranes, the impedance of the protein can be modeled as a resistor in parallel with a capacitor. Using EIS, the frequency response of this model can be found. The amplitude of the returning This information is determined by sweeping the frequency of a sinusoidal input and recording the relative amplitude and phase of the output to create a Bode Plot.
**Introduction:**

The goal of this project was to create a low cost alternative to commercially available equipment for electrochemical testing. The device must be capable of performing various electrochemical tests including cyclic voltammetry and chronoamperometry. This design minimizes the time required for analysis by providing the user instant feedback through a LabVIEW interface. This eliminates the need to send the solutions or results back to a laboratory for analysis. The wireless Bluetooth connection provides mobility, meaning that in a laboratory setting, there is no central workstation where all the testing must be performed. Instead, experiments may remain stationary because the testing equipment can be easily relocated to the testing area. This minimizes the chance of compromising sensitive chemical reactions. Further, the end product is battery powered, which means that this device can easily leave the lab and perform tests anywhere.

Current lab equipment such as the Hokuto Denko Potentiostat /Galvanostat, with Wave Generator (shown to the right from www.nadascientific.com) requires a wall outlet for power, therefore limiting the places where tests can be performed. This is a problem because there is a need to take measurements in the field. In addition to being limited to laboratory settings, these current devices cost more than $6000. This prevents many potential users from purchasing such a device.

Design team two created a device that is both portable and cost effective. The significant reduction in cost will allow more facilities to purchase a device to perform electrochemical tests. The device created by design team two could easily be mass produced for less than one hundred dollars. The minimal price-point of this device will allow for a larger range of consumers including those who do not have access to commercially available devices, or the budget to purchase one. With this larger user base, more electrochemical tests can be done which will improve the safety of consumer products.
The fast diagram above provided design team two with the insight need to set initial project goals and begin to structure work efforts. As the semester progressed, each of the above boxes were treated as a separate systems or development. This diagram is correct for with respect to the final design.
The house of quality was a great insight into the design requirements that are crucial to the success of this design project. The top three Critical Customer Requirements in order are: battery powered, reduced cost, and portability. The device created by design team two met all of these requirements.
Solution Selection Matrix

<table>
<thead>
<tr>
<th>Engineering criteria</th>
<th>Importance</th>
<th>Possible solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>On-Device Analysis</td>
<td>LabVIEW Based Analysis</td>
</tr>
<tr>
<td>Size</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Handheld Functions</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>Low Cost</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Appearance</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Low Power Consumption</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Reliable</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>Mobile</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Post-processing upgrades</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Data Storage</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>208</strong></td>
<td><strong>282</strong></td>
</tr>
</tbody>
</table>

Table 3: Solution Selection Matrix

dsPIC30F2020
The microprocessor was chosen due to its ability to digitally sample an analog signal and perform serial communication over the UART port. Additionally this microprocessor costs less than 5 dollars and takes up very little space on a PCB.

DAC
The ability to convert a digital signal to an analog potential was a necessity. Two digital to analog converters (DAC) were purchased and tested throughout the designing process. The first was the 8-bit Analog Devices AD7303 which operates from a supply of 5 volts, thus giving us 19.53 mV steps over the 5 volt supply range. This device was selected because of its low power consumption (12 mW) and DIP packaging. The second DAC that was purchased was the 12-bit Linear Technologies LTC1257 which also operates from a supply voltage of 5 volts. This was chosen because of its more precise step size of 1.22 mV over the 5 volt range. This was ultimately discarded due to the noise it introduced into the system from internal digital switching.

Digital Pot
Having an adjustable gain in the potentiostat meant that a controllable digital potentiometer was necessary. The Microchip MCP42100 digital pot provides 8-bits of resolution over the 100 kohm range, corresponding to a step size of 391 ohms. The device itself consumes only 5 μW of power and operates from a single sided supply ranging from 2.7-5.5 volts. The 14 pin, dual potentiometer package allows both devices
to be controlled individually through one data pin and one chip select pin from the microprocessor. Both sides of the digital pot can be matched within less than 1 percent, which makes this particular package ideal for the differential gain amplifiers in the potentiostat.

**OP Amps**
The Analog Devices AD8641 op amp operates from a single sided 5 volt supply making it ideal for this project. Given the need for an analog ground at 2.5 volts referenced to the power supply, the rail to rail output feature of this op amp provides maximum amplification ability. They only consume 1.25 mW of power and provide a maximum input bias current of 1 pA. This was necessary for the first design iteration that required femtoamp sensitivity. The design requirements redefined mid-semester and the femto amp was deemed unnecessary. These op amps were still used due to their small size and power requirements.

**DC-DC converters/Voltage regulator**
The Lt1300 dc-dc converter operates was selected because it can operate from two AA batteries in series with a dropout voltage of 1.8 volts. This part can output either 3.3 volts or 5 volts depending on the external connections. This was necessary because Bluetooth operates only at 3.3 volts and the rest of the system was required to operate at 5 volts by the sponsor.

**Bluetooth**
Bluetooth was a necessity due to the fact that the device is meant to be mobile. The original design incorporated the RS-232 serial connection to the computer. This hindered mobility, required extra parts that consumed large amounts of power, and increased the overall size of the project. The Parani-ESD 210 OEM Bluetooth Serial Module was used as a direct replacement for the RS-232 connection and minimized components. The Bluetooth module was necessary due to the lack of an RS-232 connection on newer laptops.

**LabVIEW**
LabVIEW was selected for the convenience of user interface development. The advantage of using LabVIEW is that the signal processing and data storage capabilities are far superior to that attainable on the microprocessor selected. The ability to process data and display it on a computer screen ultimately results in a reduction on power consumption in the system. The graphical interface available makes LabVIEW both easy to understand and appear visually appealing. The debug mode in LabVIEW simplifies programming and troubleshooting by providing the user with visual feedback. Additionally, Bluetooth can be configured easily using the LabVIEW interface.
### Budget for Prototype

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
<th>Quantity</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 pin PIC</td>
<td>$4.70</td>
<td>1</td>
<td>$4.70</td>
</tr>
<tr>
<td>100KΩ Digital Potentiometer</td>
<td>$1.69</td>
<td>1</td>
<td>$1.69</td>
</tr>
<tr>
<td>AD8641 Op-Amp</td>
<td>$3.64</td>
<td>5</td>
<td>$18.20</td>
</tr>
<tr>
<td>PARANI-ESD210-01 Serial-Bluetooth Module</td>
<td>$57.95</td>
<td>1</td>
<td>$57.95</td>
</tr>
<tr>
<td>DC/DC UP CONVERTER</td>
<td>$5.00</td>
<td>2</td>
<td>$10.00</td>
</tr>
<tr>
<td>CTX10-1-R - Power Inductor</td>
<td>$5.25</td>
<td>2</td>
<td>$10.50</td>
</tr>
<tr>
<td>1N5817 - Schottky Diode</td>
<td>$0.37</td>
<td>2</td>
<td>$0.74</td>
</tr>
<tr>
<td>Project Case</td>
<td>$9.54</td>
<td>1</td>
<td>$9.54</td>
</tr>
<tr>
<td>8-bit D/A Converter</td>
<td>$5.60</td>
<td>1</td>
<td>$5.60</td>
</tr>
<tr>
<td><strong>Total Project Cost</strong></td>
<td></td>
<td></td>
<td><strong>$118.92</strong></td>
</tr>
</tbody>
</table>
Chapter 3

Potentiostat

The final design of the potentiostat is shown below. The final gain and element choices are shown below.
\[ R_{pot1} = R_{pot2} = R_{pot} \]
\[ R_2 = R_3 = R \]
\[ V_{out} = \frac{R_{pot}}{R} (I_{we} R_1) \]

The current coming into the working electrode is first converted to a voltage and amplified by the value of the resistor \( R_1 \). Next, the circuit was designed to subtract out the applied voltage \( V_a \) using the differential amplifier. This differential amplifier also provides a gain of \( R_{pot}/R \). See Appendix III for the PSpice simulation of the current readout amplifier circuit.

The nominal component values used in the final design are shown below.

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>100</td>
<td>( \Omega )</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>1</td>
<td>k( \Omega )</td>
</tr>
<tr>
<td>( R_3 )</td>
<td>1</td>
<td>k( \Omega )</td>
</tr>
<tr>
<td>( R_{pot1} )</td>
<td>100</td>
<td>k( \Omega )</td>
</tr>
<tr>
<td>( R_{pot2} )</td>
<td>100</td>
<td>k( \Omega )</td>
</tr>
<tr>
<td>( R_4 )</td>
<td>10</td>
<td>k( \Omega )</td>
</tr>
<tr>
<td>( R_5 )</td>
<td>10</td>
<td>k( \Omega )</td>
</tr>
<tr>
<td>( R_6 )</td>
<td>10</td>
<td>k( \Omega )</td>
</tr>
</tbody>
</table>

All components were tested and found to be within 1% of all these values.

The triangle wave shown (bottom) is the applied voltage for cyclic voltammetry. In order to simplify viewing of different waves the current coming into the working electrode is modeled as a 1 Ampere since wave (top). This is only done to simplify gain calculations as well as visually distinguish the current from the applied voltage signal. It is seen that the voltage at \( V_{out} \) (middle) wave with no addition of the applied triangle wave. This means that the input is perfectly canceled out during the analog amplification. Also the magnitude of \( V_{out} \) is 10kV. This corresponds to a total gain of 10,000 or 80dB.

This gain will allow the user to analyze responses and obtain results at 10-bit resolution on signals of 500\( \mu \)A\textsubscript{p-p} or higher. Since the gain is controlled by the digital potentiometers, and the op-amps are stable at unity gain, it is possible to reduce the gain such that responses as large as 50mA\textsubscript{p-p} can be captured at 10-bit resolution. Any response signal greater than 50mA\textsubscript{p-p}, will cause the op-amp outputs to be clipped.
Microprocessor

The microprocessor is crucial to the implementation of this project. All of the required voltage signals are generated on the microprocessor as well as controlling the communication to the other digital devices. The dsPIC30F2020 is a 28 pin DIP with 1 Universal Asynchronous Receiver/Transmitter (UART) port. We are using the Microchip MPLAB application to program the microprocessor over the UART pins 18 and 19. It is physically connected to the computer through the MPLAB ICD2 debugger. This allows any computer with the MPLAB program and a USB port to program this microprocessor. The only disadvantage to using the ICD2 is that the microprocessor can not utilize the debug feature and can only operate in programming mode. This UART port is also used to communicate with the computer via the Bluetooth module. Communication can only occur if both the computer interface and the microprocessor have the same baud rate, specified by the U1BRG register on the PIC. This can be calculated with the following equation:

\[
U1BRG = \frac{\mathrm{FCY}}{16 \times \text{BaudRate}} - 1
\]

For this project, a baud rate of 9600 was required. The operating mode is FRC_HI_RANGE which corresponds to a clock cycle frequency of 19.7 MHz.

\[
U1BRG = \frac{19.7\text{MHz}}{16 \times 9600} - 1 = 127.25
\]

Therefore, the integer value 127 should be assigned to this register, thus enabling successful communication between the microprocessor and computer. The next step is to set all the necessary configuration bits for receiving and transmitting interrupts:

```c
U1MODEbits_UARTEN = 1; // enable UART
U1MODEbits_STSEL = 0; // 1 stop bit to end transmission
U1MODEbits_PDSEL = 000; // 0 parity bits
U1MODEbits_BRGH = 0; // High Baud Rate Enable bit (set standard)
U1MODEbits_RXINV = 0; // 1 is idle bit
U1MODEbits_LPBACK = 0; // Disable loopback mode
U1MODEbits_USIDL = 0; // Continue UART in idle mode
U1MODEbits_ALTIO = 0; // Use U1TX and U1RX not U1ATX and U1ARX
U1MODEbits_IREN = 0; // Disable IrDA encoder and decoder
U1BRG = 127; // 9600 baud rate
U1MODEbits_ABAUD = 0;
U1STAbits_UTXEN = 1; // Enable transmission
IEC0bits_U1TXIE = 1; // Enable transmit interrupts
U1STAbits_UTXBRK = 0; // Disabling break sequence
```
U1STAbits.UTXISEL1 = 0; //Interrupt generated when any character is
//transferred to the Transmit Shift Register
U1STAbits.UTXISEL0 = 0;
IEC0bits.U1RXIE = 1;   //UART interrupt receiving is enabled
IFS0bits.U1RXIF = 0;   //initialize reciever to 0
IFS0bits.U1TXIF = 0;   //initialize transmitter to 0
U1STAbits.URXISEL = 0b00; //Interrupt whenever a character is received

In order to test transmitting and receiving data, we initially connected from the
computer to the microprocessor using the RS-232 port and a LabVIEW interface.
Whenever a character is sent from LabVIEW to the microprocessor, the receive buffer
U1RXREG becomes full which sets the U1RXIF flag to a 1 and an Interrupt Service
Routine (ISR) is initiated. Inside this function, the data stored in the U1RXREG buffer can
be read by storing its value to a global variable:

void __attribute__ ((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{
    if(U1STAbits.OERR==1)
    {
        U1STAbits.OERR = 0;
        return;
    }
    while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;  //store the received data to a global variable
    IFS0bits.U1RXIF = 0; //reset the interupt flag
}

The program will resume if the U1RXIF flag is cleared and it reaches the end of the ISR.
This received data can be used to specify which test the user would like to run and the
parameters that go along with the test. Similarly data can be sent to the computer
through a transmit interrupt. This occurs when a user writes data to the transmit buffer
U1TXREG. This sets the U1TXIF flag to 1 and goes to the ISR for the transmit interrupt.
The data that has been written to the transmit buffer gets sent through the UART port
17 on the PIC to the computer. The sending and receiving of data between the
microprocessor and the computer is all taken care of behind the scenes. This allows the
user interfacing with the device to simply select the test and its corresponding
parameters and receive the results. Switching from the RS-232 connection to the
Bluetooth module was a fairly easy change, because the Bluetooth utilizes serial
transmission. The module needs to be sent AT commands or initialization commands to
in order to activate the connection (Appendix III: Source Code). The first AT command
tells the Bluetooth module not to send verification back to the microprocessor. The
second command tells the module to enter mode 3 or discovery mode until a
connection is established. The third command is a software reset, which had to be
enabled in order to set the device in discovery mode.
The microprocessor can only generate digital signals. In order to generate an analog signal to be sent to the potentiostat and ultimately the solution, the microprocessor must interface with a digital to analog converter (DAC). The chip select pin must be set low in order to initialize data transfer. The data is latched on the falling edge of the DAC’s clock. This clock is separate from the microprocessor’s internal clock and is manually set high and low to avoid timing issues. To end data transmission, the chip select pin must be set high. The implementation of code that controls the DAC can be found in (Appendix III: Source Code). Similarly, the digital potentiometer can be controlled by the microprocessor using the same techniques as the DAC.

The response from the solution is an analog signal, which must be sampled and converted to a digital signal in order to send the data to the computer. This analog to digital conversion is done by the microprocessor, which has the capability of sampling with 10 bit resolution but serial data transfer can only send 8-bit chunks at a time. Transferring the full 10 bits of resolution involves splitting the sampled data into two 8-bit chunks with the 2 most significant bits shifted to a new byte. The microprocessor would send these two bytes of data separately to be reconstructed by the computer and thus taking advantage of the full accuracy. The microprocessor is connected so that pins 2 and 3 are analog inputs to be sampled. In order for the microprocessor to recognize pins 2 and 3 as analog inputs, the TRISB register should be set to 0x03. The rest of the configuration settings can be found in (Appendix III: Note 1 and Appendix III: Source Code).

The entire program used for this device can be found in Appendix III: PIC Code.
LabVIEW

The LabVIEW programming takes care of the whole machine to user interface system as well as processing data. The user will only see the front panel of the LabVIEW program. On the front panel there is a drop down menu that is used to configure the type of communication between LabVIEW and the rest of the system. This is an automatically generated list of communication ports that can be chosen by the user. The Bluetooth connection can easily be enabled by selecting the COM40 port in the LabVIEW dropdown menu. Here test controls, parameter selection, graphs, and export data types are seen. The user has the ability to choose which test they want to run, chronoamperometry or cyclic voltammetry.

Inside the chronoamperometry test the user sets the parameters of start and stop voltages. Once the start test button is pressed the program will run the user defined test, and output the generated and response graphs for the user to see (example of this see Appendix III) The user then has the option to export the data to excel, word, notepad, etc. For the cyclic voltammetry test, the user can choose between two different waveforms, saw tooth and triangle. Scan rate, peak-to-peak amplitude, DC offset, number of cycles, and delay cycles before data is transmitted from the PIC to LabVIEW can also be set by the user. The program has defined limits for the inputs that are acceptable, and will not allow the user to exceed these limitations. Also the cyclic voltammetry test allows the user adjust the gain to optimize the resolution of test results. When the user selects the run test button all the parameters are sent to the microprocessor. The user will then see two graphs as outputs, an amplitude vs. time graph and a hysteresis graph (example of this see Chapter 4). In the two graphs the user will observe the signal sent into the system and the response from the system on the amplitude vs. time graph. On the second graph the user sees a hysteresis curve that was made by graphing the response current vs. the input voltage. The user has the option to either save the data (in excel, word, notepad, etc) or discard the data and run another test. There is a third test called Electrochemical Impedance Spectroscopy (EIS). From an EIS test the user can see the response of the system to a range of frequencies, 10 mHz to 1 KHz. The EIS test shows the theoretical operation of a frequency sweep on a solution. In the program the EIS test runs a simulation of data created by a PSPICE. The simulation is based upon a circuit model for a cellular protein. In the program the user can see both the magnitude and phase portions of a Bode plot.

Behind the LabVIEW user interface is the block diagram implementation of the user separate tests. These are created by using a VISA read and write sequence. The first block in the sequence is the VISA Serial block. Here the program specifies communication parameters like baud rate, data bits, termination character enable, termination character, stop bits, time out, and VISA resource name. The output of this block is a VISA resource name and an error out. Next in the block diagram is a button that allows the user to select which test to run. When this is pressed on the front panel the specified test will run. Incorporated within this button is the default start for when no tests are running, such as before or after tests are completed. The next step of the sequence deals with a case structure in LabVIEW. A case structure allows for different operations based upon a selector. The selector for the case structure used in this
program is the user defined test. Based upon what test the user chooses, different variables will be passed into the system. For example, in the cyclic voltammetry test only the variables used in cyclic voltammetry (scan rate, peak-to-peak voltage) will be passed into the program case structure. The purpose of the first case structure in the program sequence is to utilize the VISA write to send the user defined test information to the microprocessor. Before the parameters are sent to the PIC, logic checks are performed on the data. These checks include comparing parameters against limitation values or summing different voltage levels to see if they exceed the design constraints of the potentiostat. If a parameter selection fails, an error message will be displayed. Once the parameters pass through the logic checking sequence they are sent to the PIC using the VISA write block.

The VISA write block takes in the VISA resource name that has been passed to it and writes the data to the specified port location. The next step in the sequence is the case structure that is used to implement VISA read. However, before the LabVIEW program can enter this case structure, there must be a time delay created by a dynamic delay loop. This delay is necessary to ensure that data is not lost in the communication process. Depending on the parameter selection by the user, the program will automatically scale the delay to the minimum time required to ensure data integrity. Once the appropriate delay is achieved, all the signals can enter the case structure that house the VISA read implementation. The VISA read will then receive information from the PIC. This information contains the generated signal and the response signal. Once LabVIEW obtains these signals, logic is implemented to transform the columns of data into a usable format to create the graphical interface. The data received is manipulated and displayed on the appropriate graph. The graph for chronoamperometry simultaneously displays the voltage vs. time for both the signal and response signal. A legend is used to distinguish the two signals. Cyclic voltammetry creates two graphs. The first displays input and response signals vs. time, while the second displays the output signal vs. input signal which shows a hysteresis curve.

Electrochemical impedance spectroscopy presents the data in a Bode plot. This display is created by measuring and recording the maximum voltage and time at which it occurred in both the input and response signals. This is repeated over a frequency range from 1 mHz to 1 kHz with 10 frequency points per decade. The Bode plot can then be constructed from the following formulas:

\[
\text{MagnitudePlot} = 20 \log \left( \frac{V_2}{V_1} \right)
\]

\[
\text{PhasePlot} = 360 \times (\frac{1}{2}) \sqrt{-t_2^2 + f}
\]

(See Appendix III, figures 6 and 7 for EIS and PSpice display comparison)

After the results are obtained, the user will be prompted on whether to save or discard the data. If the user chooses the discard data the program will return to its default case state, waiting for the next test parameters. If the user chooses to export the data, a window will automatically pop up asking for a location in which to save the data file.
The data is formatted and saved into two columns automatically by the LabVIEW program. The user can save the data as an excel file, text document, or similar format. The case structure will terminate upon successful completion of discarding or saving the data. The sequence will then complete by entering the VISA close block, which terminates the VISA communication link. This sequence runs repeatedly using a while loop. The whole sequence resets itself at the end of every case, returning to the default state awaiting parameter commands from the user.

Reference to Appendix III figures 2, 3, 4 for visual of LabVIEW front panels and 2a, 3a, 4a for the block diagrams as well as example of use.

**EAGLE**

EAGLE is the software tool that was used to create the PCB the project. In EAGLE, the circuit diagram must be created first. Many of the parts needed to complete the circuit layout can be found in the default libraries. Some of the parts that were used in this design could not be found in the standard EAGLE parts library. However, EAGLE allows for the option to design a library from scratch. Two parts were constructed using this method: the operational amplifiers used in the potentiostat design and the inductors used in the power system. To design custom parts libraries a new window must be opened for the part editing tool. Here both the circuit diagram and PCB part layout can be created, as well as assigning the individual pin layouts. After completion of this step, the library can then be added and used in the regular EAGLE program. Once all the components for the whole project schematic were located, the circuit layout could then be created. This yielded a circuit diagram from which to work. Next in EAGLE, the PCB layout was completed. In this window, all components are shown to scale. The desired board size and shape must be drafted before the circuit elements can be distributed. Due to the large number of components and the desire to minimize project size, this presented quite a challenge. In the EAGLE PCB view, the auto routing feature was used to assist in initial trace layout. In the end, the majority of traces were manually routed to successfully design the board. Only two conducting planes were used due to the limitations of the ECESHOP, which again limited the amount of traces and routes that can be made. Upon successful completion of these steps, the board was then saved in extended Gerber 274 format and sent to the ECESHOP for milling, where a physical board will be produced.

**Microprocessor**

The microprocessor has a maximum data SRM of 512 Bytes from figure 2. Towards the end of the semester, there were lots of extra features that were not necessary in the final implementation and were pushing this capacity to the limit. Operating near the maximum of this data space was yielding very strange and undesirable results. Through
the optimization of variables and management of functions, the microprocessor cut its
data SRAM usage down considerably.

| Product               | Pins | Packaging | Program Memory (Bytes) | Data SRAM (Bytes) | Capture | Compare | UART | SPI | I²C™ | PWM | ADCs | S & H | A/D | Input | Analog | Comparators | GPIO |
|-----------------------|------|-----------|------------------------|-------------------|---------|---------|------|-----|-----|-----|------|------|------|------|------|--------|--------|--------|
| dsPIC30F2020          | 28   | SDIP      | 16K                    | 512               | 3       | 1       | 1    | 1   | 1   | 1   | 4x2  | 1     | 5    | 8    | 4     | 21     |

When the microprocessor is operating, the system can draw anywhere from 500-800 mA. This large current draw drains the battery supply very quickly. This was remedied through the use of sleep mode. In this mode, the battery only drains 40-120 mA, which is a serious reduction in power consumption.

At the beginning of the project, the device was connected to a computer via an RS-232 connection. The specifications of the project as given by the sponsor were to eradicate this limitation. The end result was a portable battery powered device that communicated wirelessly to the computer via a Bluetooth connection.

The signal generation for the electrochemical tests was very clean when directly connected to the power supplies. However, noise was introduced to the system when connected to the battery through a dc-dc converter. This noise was minimized by connecting capacitors across the power supplies. Also an inductor with a damping diode connected in parallel with it was added from the output of the DC to DC converter to help reduce the noise. When laying out the power system to the board, design considerations were taken to minimize the noise. The project design uses the LT1300 step up DC to DC converter. The first design consideration is to keep the high input capacitor no more than 3 inches from ground and V_{in}. The next design consideration is to run the trace from ground to PGND under the board directly from pin 8 to pin 1. Also keep the switch pin as short as possible. All this will help to reduce the amount of noise in the system.

The PCB layout designed had two parts, the potentiostat and the digital implementation. Each of these parts were put onto their own PCB board, and the box that they were to fit into is small. So consequentially, one board was attached to the top of the case and the other was attached to the bottom of the case. When these two halves got put back together, there was not enough space for the components. To get the head room for the components a dremel was used to lower the stand offs. This made the inside of our small box have more space in between the two PCB layers.

The PCB layout designed had two parts, the potentiostat and the digital implementation. Each of these parts were put onto their own PCB board, and the box that they were to fit into is small. So consequentially, one board was attached to the
top of the case and the other was attached to the bottom of the case. When these two halves got put back together, there was not enough space for the components. To get the head room for the components a dremel was used to lower the stand offs. This made the inside of our small box have more space in between the two PCB layers.

**Power Layout**

The Power layout for the system consists of two DC to DC converters that are fed from two double AA batteries. The two DC to DC converters send out voltage levels of 3.3V and 5V. This is done by the internal working of the converters with the help of external components. Below you can see the component layout of the implementation of the DC to DC converter that is being used.

![Two-Cell to 3.3V/5V Step-Up Converter](image)

DC to DC converter configuration used in the design Project

For the inductor L1, the use of one side of a transformer was implemented. The use of both forms of output was used in the design project. To do this all that was needed was to set the selector pin to high or ground. When grounded the output is 3.3V and when set to high the output is 5V. Also the design used the current limiter function of the DC to DC converter. This helps protect the circuit during startup, just in case of a huge startup spike in current. The two input AA batteries operate a about 3V when fresh, but as use goes on the voltage decreases. This system of DC to DC conversion allows for the voltage to drop all the way down to 1.8 V. In the entire voltage range of three volts to 1.8 volts the voltage can be regulated to either 3.3V of 5V.
Numerous tests were performed with the device prototype. These tests were conducted on various concentrations of Potassium-Ferro cyanide solution or even Mt. Dew. The results are shown below.

This test represents 100mM Potassium-Ferro cyanide solution tested using cyclic voltammetry at a scan rate of 900mV per second.
This test represents 10mM Potassium-Ferrocyanide solution tested using cyclic voltammetry at a scan rate of 500mV per second with a DC offset of -200 mV.

This test represents extremely oxidized 100mM Potassium-Ferrocynide solution tested using cyclic voltammetry at a scan rate of 500mV per second with a DC offset of 1000mV. Using a larger offset yields differences in the hysteresis plots, however the
fact that this solution has been reacting with the air for long periods of time has created a completely different shape of hysteresis curve.

This figure represents Chronoamperometry with a starting voltage of -1500mV stepping up to a final voltage of 200mV using 100mM Potassium-Ferrocynide solution.
The design of the electrochemical sensor was a success. The product prototype cost just less than $120. When put into mass production, the cost per-unit is estimated to be only $60; far less than the required $100. This product was created with a total development cost of $580 not including labor. This is slightly higher than budget allocated to the design teams at the start of the semester. However, this figure does include enough parts to construct a second prototype.

The final design was completed as schedule for presentation on Design Day, December 11, 2009. Deviations were made from the original project schedule as needed to accommodate dynamic project requirements. Due to sound project management, the
original plan included strategic slack time and therefore there was no detriment to the delivery date.

While the project successfully fulfilled many of the Critical Customer Requirements there is room for improvement. Due to limited manufacturing resources provided to the design teams many integrated circuit chips are in a larger package (DIP), which caused the project to be larger than necessary. Another side effect of this increased size is increased power consumption. If smaller surface mount (SOIC) ICs where used, the size and power consumption would be reduced. In addition, a multi-layer PCB would allow for much more compact circuit layout. All of these improvements would result in a smaller and therefore more portable product.

Also, numerous other electrochemical tests are possible to implement on this device with a simple software update. These as well as the current tests would benefit from a higher resolution digital to analog converter. This only requires that one chip be swapped out with some re-routing of traces and a minor update to the code be completed.

Due to time and physical space limitation the electrochemical impedance spectroscopy testing circuitry was not included. However, the LabVIEW user interface is finished and included. In order to complete the test an external monolithic function generator would likely need to be used. This will require that a large, switch controlled capacitor and resistor network be implemented. This network will provide a selectable RC time constant required to control the frequency of the function.
Conclusion
The final product includes the functionality to perform chronoamperometry and cyclic voltammetry. These electrochemical tests can be performed in the field due to the device’s small size and wireless Bluetooth connection. Each test allows for highly controllable user-selectable parameters that are used for customized testing. Given the extremely low cost of the device this product can be easily acquired by facilities that would perform electrochemical testing. This device will open up new markets for low cost electrochemical testing. This testing can benefit people in areas such as medical testing and food safety.

Appendix I

Luke Lapointe

My technical contribution to this project includes programming and debugging the microprocessor, dc-dc power conversion, serial communication through the RS-232 port and Bluetooth, LabVIEW programming and debugging, analog to digital conversion, programming a digital potentiometer, and general debugging issues.

The microprocessor is really the heart of this project. It generates all of the voltage signals and handles all the communication between the computer and our device through a serial connection. We initially used an RS-232 corded connection to interface with the computer. This connection required us to step up our voltage from 5 to 12 volts. For this DC to DC step up in voltage I used the MAX232 which can step up voltages from a single 5 volt supply using a capacitive voltage generator. In order to communicate serially, both the PIC and the computer must have the same baud rate as well as other configuration settings. I was heavily involved in configuring the PIC to receive and transmit interrupts over the UART port. This essentially allowed us to send data back and forth from the PIC to the computer. Later in the project, a heavy emphasis was placed on being wireless and handheld, which is ultimately what drove us to replace the corded RS-232 connection with a wireless Bluetooth connection. All of the required voltage signal tests were done on the microprocessor. I helped code the cyclic voltammetry signal generation on the PIC, which is a triangle or saw tooth wave with adjustable scan rate, amplitude, and dc offset.
The other test that I helped program is chronoamperometry, which is a dc step in voltage where the user can select the starting and ending voltages. These tests were sent digitally through an external 8-bit digital to analog converter to the potentiostat where the solution is connected. The response is sent back through the potentiostat and sampled by the microprocessor. I was heavily involved in configuring the analog to digital conversion on the microprocessor, as well as utilizing the full 10-bits of sampling. I wrote the functionality to split the 10-bit data into 2 bytes to be read by the computer, which greatly smoothed out the signal. LabVIEW was the preferred computer interface and I helped write and debug several functions, including reading in the 10-bit sampled data. I also helped write a function on LabVIEW that would enable the user to adjust the gain of the potentiostat by controlling a digital potentiometer. This involved interfacing with an external 100 kilo ohm digital pot with 256 discrete levels, which corresponds to 390 ohm steps in resistance. The timing between the microprocessor’s clock and writing data to the digital pot was not 1 to 1, so I had to manually set the digital pot’s clock high, write the data to be latched on the falling edge, and then set the clock low in order to successfully utilize the digital pot. Towards the end of the semester, I became heavily involved in the debugging of all aspects of our project and ensuring that everything could communicate and function properly.

Nick Timpf
My main portion of the project was microprocessor programming. I also helped with the initial research and design of the system, as well as picking of components for the system. Furthermore I helped to streamline and implement the LABVIEW user interface. In conjunction with the coding portion of the project I worked on the analog circuitry. I was involved a lot in the initial building, testing, and troubleshooting of the potentiostat. A lot of this was double checking my teammates work on the wiring of the circuit, which some of those late nights was very necessary, along with figuring out if what we were seeing the potentiostat do was correct. My work on this project also branched out into the design and creation of the power system for the project. As our project is portable, it runs on batteries which require voltage regulation and boosting of the battery voltage. One of the first tasks in the project I did was the setup and wiring of the different components, such as the digital potentiometer, digital to analog convertor, and Bluetooth module to the microprocessor. The digital potentiometer was used to implement a variable gain of the potentiostat, the digital to analog convertor was used to create the different signals in order to perform the different electro-chemical tests, and the Bluetooth made our device very portable being completely wireless.
The setting up and wiring of the components involved a lot of data sheet reading in order to find out what each pin needs to be sent from the microprocessor in order to control the chip as desired. Once the wiring was done I worked on the generation of different voltage waveforms that are used for the different tests. These waveforms were generated using the digital to analog convertor however once the digital to analog convertor code was implemented the generation of signals was very easy due to good flexible coding of the convertor. The types of signals that we generate are triangle and linear sweep for cyclic voltammetry and a step input for chronoamperometry. The signal that proved to be the most difficult was the triangle wave as it had to have a variable scanrate. This took many hours to test and verify as a 2V peak to peak triangle wave at 10mV/s takes around 6 minutes to generate a full period of the signal.

I also coded in LABVIEW, I did not work on the general creation of the code and interface but I did code a lot of the fine tune things such as the implementation of the linear sweep wave that was not originally part of our design specifications. I also coded a little bit of the interface here and there, mostly just when changing a minor aspect of how the code was implemented due to a change in the microprocessor code.

On top of all of these things, I also was the second most knowledgeable person regarding the potentiostat. I did not do any of the building of it nor did I directly design it. What I did for the potentiostat was to be the guy to bounce ideas off of as well as talk through what is happening in the potentiostat. This proved useful in the testing of the potentiostat as we knew what we were expecting to see in the response of the system.

Lastly I helped to test and debug the power system. Our power system proved to be quite temperamental at first. There was a lot of debug that happened to it in order to get it to output currents that were just half of the rated output current of the device. In the end it took a couple optional components on a current limiting pin of the device as well as several capacitors and even an output inductor and diode to get the system to deliver relatively clean and reliable power.

Mark VanCamp

The specific area I was responsible for in our project was that of PCB design. The need for a PCB was not needed until later in the semester, so I have been helping with other areas of our design. I have primarily been working on the LabVIEW interface with Steve. I first created an interface for the three different tests chronoamperometry, cyclic voltammetry, and electro impedance spectroscopy. The way I initially accomplished this was by creating an excel file that represented serial data that would be sent from the microprocessor. I then had LabVIEW take in this created “serial data” and separated the current and voltage values from the time values they occurred. Once I broke up this data stream I was able to make the appropriate graphs for each
test respectively. The next process that I worked on was getting LabVIEW to read in a serial connection. This proved to be a slightly challenging task. I read through many online LabVIEW tutorials to learn how the baud rate and other similar parameters needed to be defined to generate the desired results in LabVIEW. Once LabVIEW could read in serial data I helped enhance the level of communication it shared with the microprocessor. What this entailed was making LabVIEW our user interface where the tests would be started and parameters such as scan rate and step levels for the different tests could be selected. Having these selections available in LabVIEW makes the controls on the unit unnecessary. However these controls could still be added at a later date if so desired by another design team. The user interface on LabVIEW also gives an option where data from the graphs could be exported to a destination selected by the user for later review.

Near the end of the semester the need for the PCB has arisen so I have been developing my understanding of EAGLE (a PCB design and layout program) and began work on designing some of the components we would need. In order to gain a better understanding of EAGLE I have read numerous tutorials and communicated with Brian Wright (ECE shop employee) on a regular basis. Once I understood how this program worked I was able to design our schematic. I worked closely with my team so I could create an accurate schematic based off of our circuits on our protoboards. Once this first schematic was created I then began the actual designing of the board that would be used in our final project. After spending numerous hours placing the components and laying out their traces it became apparent to me that two printed circuit boards were necessary (due to small size of the project box we selected for final design). Next I created a second schematic and designed another board where the remaining components and traces could fit reasonably. The ECE shop has milled both of these boards at no cost to our team. I then took the boards to the machine shop in the engineering building and had them cut the PCBs to length so they would fit in our project box. Finally I helped assemble our circuit on these boards and mounted them in our case.

Brent Woodman
At the beginning of the project I heavily researched applications and techniques of functional proteomics. This was simply to gain a deeper understanding of electrochemical testing; what is done and what information is obtained from this testing. I was later responsible for the amplifier design as well as the choice of components. I designed and tested numerous configurations for our amplification circuitry. At the start of the project, it was required that the circuitry provide a total gain capable of resolving femtoamp magnitude response currents. This required extremely precise techniques for biasing amplifiers in order to cascade many stages. The biasing was a
huge problem since the total gain of the system was on the order of 1,000,000,000,000 or approximately 260 dB. At this level, any slight DC offset in the input resulted in the output hitting one of the power supply rails. I completed numerous PSpice simulations as a way of proving that. However, after revisiting the specifications of the project, many of the looming technical challenges were found to be beyond the scope of this project. For the overall project, I worked with the rest of the team to determine our best course of action when each decision was made. Once it was decided that our project would be a product that utilized a computer interface we split into sub groups. My task in the sub group was to create the AC readout in LabVIEW. This was done by defining the way in which calculations were made for electro-impedance spectroscopy. On top of this I created test files for the AC readout using the PSPICE simulations of a protein model. Lastly I aided in the troubleshooting of our DC-DC convertors in the power system of the project. The convertors were not outputting a constant voltage as they should and used various techniques learned in analog circuitry classes to stabilize the output voltage of the system. Part of this stabilization problem was due to switching noise in the system caused by the DC-DC convertor. I used a filter to drastically minimize this noise.

Steve Zuraski
My technical responsibilities as stated in our team proposal are to implement the LabVIEW programming. The LabVIEW programming takes care of the machine to user interface. The goal of the user interface was to make it as user friendly as possible. The front panel that the user sees is very simple. There are the input parameters that the user can choose. These will define the test that they also choose. There are large easy to follow buttons that will start the testing. Also there are options for the user after the test to either save the data into any file type they want, like excel word or notepad, or discard the data from the test. The LabVIEW program I created contains a VISA read and VISA write data transfer sequence that is implemented over a Bluetooth connection. The program starts with the user choosing the test and test parameters from the front panel. The front panel in LabVIEW is all the user will see. The front panel also contains the end result, the graphs that will show the signal sent into the solution and the response signal from the solution. In addition the front panel has a button called get data. This button will automatically take the data from the test and export it to the user desired location. When the user chooses this button a prompt will automatically pop up and the user can then choose the desired location of the data, like an excel spreadsheet. Also in LabVIEW there is the block diagram. The block
The block diagram that I created contains the VISA read and VISA write sequence inside of a while loop. The while loop allows the system to reset itself after a test, so the whole system does not need to be turned off in between each test. The first step of the sequence is the VISA write case structure block. I use case structure because each test has different parameters that need to be sent to the microcontroller. The VISA write block will take the user defined parameters, check to see if they are within the limits of our system, and then put the data in the correct form so it can be read by the microcontroller. Once the sequence leaves the VISA write block, it goes into a delay loop. The delay loop is dynamic to each test and the parameters within each test. It is necessary to have a delay loop between write and read so that the data doesn’t get lost or altered in transfer. Then the last phase of the sequence is the VISA read block. This also utilizes a case structure because each test has different data to be read. In the VISA read block the data is gathered in one long string. The data is then sorted in the appropriate manner depending on the test and then converted into the correct form to be displayed on a graph. The user sees the graph on the front panel. Also in the VISA read block there is the logic behind the get data button. The get data button when pressed will present the user with an option for where to save the data. The data is determined to be saved in columns that are delimited by spaces. Once the data sequence leaves the VISA read block it enters an end VISA block. This ends the chosen test and returns the program to the default state that is waiting for more user inputs.

The power to device was another technical role I implemented. The outcome was ordering a couple of DC to DC converters to regulate the voltage from two batteries to voltage levels of 3.3V and 5V. There were also external controls that had to be chosen to make the DC to DC converters. As a team we chose the inductors, diodes and capacitors to be used in this power system.

I created the PCB layout for our final device with collaboration from Mark VanCamp using the Eagle software package. We started by identifying all the components. Some of the components in our device were not in the parts library, so we had to create our own part. Once we created our part we added to the Eagle library. Next we created the circuit diagram. From the circuit diagram eagle places all the parts below a PCB area, and one can place all the parts into the design area. We ended up creating two printed circuit boards, one for the potentiostat and one for the digital signals and power. These both ended up being complex printed circuit boards due to the small size constraints of our final device. Once our printed circuit boards were designed they were sent to the shop to be milled out. Once they were milled we placed all of our components onto the board and tested it. See Appendix III figures for references to the LabVIEW program, power circuit diagrams, and PCB board layouts.
Appendix II

Appendix 2 – Literature and website references


Appendix III

Figure 1. PSpice simulation of Potentiostat Amplifier

Potentiostat Trial
Vpic 1 1B Pulse(-1 1 0.001 .2 .2 0.001 .4002)
Vdc 1B 1A -1
Vagnd 1A 0 2.5
Iwe 6 0 Sin(0 1 5)
X1 1 6 5 OPAMP
R1 6 5 100
R2 5 4 1K
R3 1 2 1K
Rpot1 100K
Rpot2 100K
X2 2 4 3 OPAMP
.subckt opamp 1 2 3
RI 1 2 100MEG
EA 3 0 1 2 100MEG
.ends opamp
.tran .001 2 0 .001
.probe
.end
Figure 2. Chronoamperometry User Interface

Figure 3. Cyclic Voltammetry User Interface
Figure 4. EIS User Interface

Figure 2a (First 1/3 of block diagram)
Figure 3a (second 1/3 of block diagram)

Figure 4a (last 1/3 of block diagram)
Figure 5. Cyclic Voltammetry Complete Output
Figure 6. LabVIEW test data from PSpice
Protein Simulation
V 1 0 AC 1
R1 1 2 100k
C1 2 3 .0075u
R2 2 3 5k
c2 3 0 10000p
.probe
.AC DEC 10 .001 1K
.END
Figure 8. PCB layout for the PIC, Power, DAC, Bluetooth, LED
Figure 9. PCB schematic lay out for PIC, Power, DAC, Bluetooth, LED

Figure 10. PCB layout for Potentiostat Circuit
Figure 11. PCB schematic for Potentiostat Circuit
Note 1: Configuring the A/D Module

The DSPIC30F2020 has an A/D module already in place but in order to use it the necessary control registers have to be set.

- **ADCON** or the A/D Control Register, which consists of 16 bits.
  - **ADON**: A/D Operating Mode bit. Setting this bit to a 1 means that the A/D converting module is enabled and 0 disables it. This bit is usually turned on after all the other configuration bits are set.
  - **ADSIDL**: Stop in Idle Mode bit. 0 means it will operate in idle mode and 1 means that A/D conversion will stop once idle mode has been entered.
  - **FORM**: Data Output Format bit. Setting this to 1 means that the output will be displayed as a fraction and 0 will display as an integer.
  - **EIE**: Early Interrupt Enable bit. Setting this to 1 will send an interrupt after the first A/D conversion is completed, whereas 0 will interrupt after the second interrupt. We will be using 0 in our project, because we want to convert two signals at the same time before sending them to the computer. This bit can only be changed when ADON is set to 0.
  - **ORDER**: Conversion Order bit. This specifies whether the even or odd analog input is converted first. Setting to a 0 means that the even input will be converted first then the odd and 1 is vice versa.
  - **SEQSAMP**: Sequential Sample Enable. Setting this bit to a 0 will means that the shared S&H is sampled concurrently with the dedicated S&H, unless the shared S&H is busy. In this case, then the shared S&H will sample once a new conversion is started. We will be using 0 for our project for this reason.
  - **ADCS<2:0>**: A/D Conversion Clock Divider Select bits. This is based on the Frequency that the A/D module is set to and whether or not PLL is enabled. We have PLL enabled and wished to have a frequency divider of $F_{ADC}/14$, so we set this bit to 5 or 101 in binary. Assuming the PIC is setup to use FRC with Hi-Range and have an un-tuned frequency of 14.55 MHz, then $F_{CY} = 14.55 \text{ MHz} \times 2 = 29.1 \text{ MHz}$ or 29.1 Million Instructions Per Second (MIPS). This means that $F_{ADC}$ should be $F_{CY} \times 8 = 232.8 \text{ Mhz}$. Thus the resulting frequency divider $F_{ADC}/14 = 232.8 \text{ MHZ}/14 = 16.6 \text{ MHZ}$ or 16.6 MIPS. This will be sufficient for our purposes. Figure 18-2 shows a derivation of the frequency of the system clock or $F_{CY}$. An alternative way to determine $F_{CY}$ is to output the PIC’s clock and measure its frequency on the oscilloscope.
• **ADSTAT** or the A/D Status Register which also has 16 bits, but only the 6 least significant bits (LSB) are usable. These bits essentially specify which pairs are ready for conversion or display the status of the conversion process. These must be cleared in the ADC interrupt after the conversion is complete.

• **ADPCFG** or the A/D Port Configuration Register is 16 bits and specifies which channels will be used as analog inputs. The DSPIC30F2020 has 12 available channels which is represented by the 12 LSB of the ADPCFG register. Setting a specific bit to 1 implies that the port pin is in Digital mode, port read input is enabled, and the A/D input multiplexor is connected to ground. A 0 implies that the port pin is in analog mode, port read input is disabled, and the A/D samples pin voltage. For our system, we want AN0 and AN1 as analog inputs, so we would set this register 1111 1100.

• **ADCPC0** or the Convert Pair Control Register #0 is a 16 bit register that controls the triggering of the A/D conversions.
  - **IRQEN0**: Interrupt Request Enable 1 bit. Setting this to 1 means that an IRQ will be generated when the requested A/D conversion from channels AN1 and AN0 is complete. For our project, we are using the AN0 and AN1 bits as analog inputs and would therefore set this bit high.
  - **PEND0**: Pending conversion of status 0 bit. When this bit is set high, that means the channels AN1 and AN0 are currently being converted. When the bit is low, conversion in these channels has completed.
  - **SWTRG0**: Software Trigger 0 bit. Setting this bit high essentially acts as a start button for A/D conversion in channels AN0 and AN1, given there are the resources available. This bit will be reset low when PEND bit is set high.


- **TRGSRC0<4:0>:** Trigger 0 Source Selection bits. This essentially lets the A/D module know where to expect the trigger to come from for channels AN1 and AN0. For our project, we will trigger off a timer 1 period match. This corresponds to 01100, but there are many other options for triggering.
```c
#include "p30f2020.h"
#include <adc10.h>

#define TIMER_PERIOD 0x9FFF // Set the timer period for 5.076 usec

_FOSC(CSW_FSCM_OFF & FRC_HI_RANGE & OSC2_IO); // Oscillator Configuration
_FOSCSEL(FRC_PLL); // Oscillator Selection
_FWDT(FWDTEN_OFF); // Turn off WatchDog Timer
_FGS(CODE_PROT_OFF); // Turn off code protect
_FPOR(PWRT_128); // Turn on power up timer 128ms
_FBS(BSS_NO_FLASH);

char pot_value[4]={'0','0','1','0'},
start_or_amp[4]={'0','0','0','0'},end_or_offset[4]={'0','0','0','0'},
scanrate[4]={'0','0','0','0'}, num_cycles[2]={'0','0'},
num_hyst[2]={'0','0'};
unsigned char Rxchar;
int V_Flag=0,C_Flag=0,Triangle_Flag=0,ind=0,DP_Flag = 0;
int data[100];
int step1[21]={10,50,100,150,200,250,300,350,400,450,500,550,600,650,700,750,800,850,900,950,1000};
long step2[21]={170000,34600,16875,10900,7975,6200,5000,4150,3550,3024,2640,2320,2043,1820,1625,1455,1311,1178,1058,958,865};
long step, pot_val=0, q = 0;
double j;

void wait(long x)
{
    long q;
    for(q=0;q<x;q++);
};

void setconbits(int j, int cont[8])
{
    int setcon_counter=0;
```
PORTBbits.RB5=0; //Set cs low
wait(j);
//set command and selectionbits
for(setcon_counter=0;setcon_counter<8;setcon_counter++)
{
    PORTBbits.RB7 = 0; //set clock low
    wait(j);
    PORTEbits.RE7=cont[setcon_counter]; //latch the control bits
    wait(j);
    PORTBbits.RB7 = 1; //set clock high
    wait(j);
}
PORTBbits.RB5=1; //set cs high thus ending data transmission
wait(j);

/**************************************************************************
*** Setting Data Bits
* Purpose: Configure and set the data bits for the DAC
***************************************************************************/
void setdatabits(int j, int bin[8])
{
    int setdata_counter=0;
    for(setdata_counter=0;setdata_counter<8;setdata_counter++)
    {
        PORTBbits.RB7 = 0; //set clock low
        wait(j);
        PORTEbits.RE7=bin[setdata_counter];//latch the data bits
        wait(j);
        PORTBbits.RB7 = 1; //set clock high
        wait(j);
    }
    PORTBbits.RB5=1; //set cs high thus ending data transmission
    wait(j);
}

/**************************************************************************
*** Digital Potentiometer Control Function
* Purpose: Set the digital pot to a specified input value
***************************************************************************/
int POT(double v)
{
    int i2=0,c12=0,modval2=1,delay=0;
    double set2=0;
    int check2 = 0;
    int bin2[8]={0,0,0,0,0,0,0,0}; //initialize data bits to 0 ohms
    int cont2[8]={1,1,0,1,1,1,1,1}; //set the control bits to set both sides
    set2=(255*v)/100000;
    /**************************
    * Round the Value
    **************************/
    modval2=1;
    check2=set2;
    if((set2-check2)>.5)
    {
        set2 = check2 + 1;
    }
else
{
    set2 = check2;
}
for(i2=0;i2<8;i2++)
{
    modval2 = 1;
    for(c12=7;c12>i2;c12--)
    {
        modval2=2*modval2;
    }
    if(set2/modval2>=1)
    {
        bin2[i2] = 1;
        set2=set2-modval2;
    }
    else
    {
        bin2[i2] = 0;
    }
}
delay=1;
int setcon_counter2=0;
int setdata_counter2=0;
PORTBbits.RB6=0; //Set cs low
wait(delay);
//set command and selection bits
for(setcon_counter2=0;setcon_counter2<8;setcon_counter2++)
{
    PORTBbits.RB7 = 0; //set clock low
    wait(delay);
    PORTEbits.RE7=cont2[setcon_counter2]; //send control bits
    wait(delay);
    PORTBbits.RB7 = 1; //set clock high
    wait(delay);
}
for(setdata_counter2=0;setdata_counter2<8;setdata_counter2++)
{
    PORTBbits.RB7 = 0; //set clock low
    wait(delay);
    PORTEbits.RE7=bin2[setdata_counter2];//send data bits
    wait(delay);
    PORTBbits.RB7 = 1; //set clock high
    wait(delay);
}
PORTBbits.RB6=1; //set cs high
wait(delay);
return 1;

/***************************************************************************/
* Digital to Analog Output Control
* Purpose: Set the digital to analog converter to a specified input value *
*******************************************************************************/
int DACA(double v)
{
    int i=0,c1=0,modval = 1;
    double set = 0;
50

int check = 0;
int bin[8] = {1,1,1,1,1,1,1,1}; //initialize data bits to max value
int cont[8] = {0,0,0,0,0,1,1,1}; //set configuration bits

set=(255*v)/5000; //calculate the number of steps neccessary to produce the requested analog voltage

/**********************
 * Round the input   *
 *********************/
modval=1;
check=set;
if((set-check)>.5)
{
    set = check + 1;
}
else
{
    set = check;
}
for(i=0;i<8;i++)
{
    modval = 1;
    for(c1=7;c1>i;c1--)
        modval=2*modval;
    if(set/modval>=1)
    {
        bin[i] = 1;
        set=set-modval;
    }
    else
    bin[i] = 0;
}
j=1;
setconbits(j,cont);
setdatabits(j,bin);
return 1;

/*************************************************************************/
* Conversion Function                                                   *
* Purpose: Convert a character array to an integer value              *
*************************************************************************/
int chartoint(char data[], int size)
{
    int count=0,temp=0,return_val=0,return_count=0,mult_val=1;
    for(count=0;count<size;count++)
    {
        temp=data[count]-48;
        mult_val=1;
        for(return_count=0;return_count<(size-1)-count;return_count++)
        {
            mult_val=mult_val*10;
        }
    }
return_val=return_val+temp*mult_val;
}
return return_val;

/*************************************************************/
*    Main Function  
*    Purpose: Driver for code 
*    *************************************************************/
int main()
{
    U1MODEbits.UARTEN = 1; //enable UART
    U1MODEbits.STSEL = 0; //1 stop bit to end transmission
    U1MODEbits.PDSEL = 000; //0 parity bits
    U1MODEbits.BRGH = 0; //High Baud Rate Enable bit(set standard)
    U1MODEbits.RXINV = 0; //1 is idle bit
    U1MODEbits.LPBACK = 0; //Disable loopback mode
    U1MODEbits.USIDL = 0; //Continue UART in idle mode
    U1MODEbits.ALTIO = 0; //Use U1TX and U1RX not U1ATX and
    U1ARX
    U1MODEbits.IREN = 0; //Disable IrDA encoder and decoder
    U1BRG = 127; //9600 baud rate
    U1MODEbits.ABAUD = 0;
    U1STAbits.UTXEN = 1; //Enable transmissions
    IEC0bits.UITXIE = 1; //Enable transmit interrupts
    U1STAbits.UTXBRK = 0; //Disabling break sequence
    U1STAbits.UTXISEL1 = 0; //Interrupt generated when any
    character is transferred to the Transmit Shift Register
    U1STAbits.UTXISLO = 0;
    IEC0bits.U1RXIE = 1; //UART interrupt receiving is enabled
    IFS0bits.U1RXIF = 0; //initialize receiver to 0
    IFS0bits.U1TXIF = 0; //initialize transmitter to 0
    U1STAbits.URXISEL = 0b00; //Interrupt whenever a character
    is received

    /***************************************************************************/
*    Make all ports outputs 
*    except AN0 and AN1    
***************************************************************************/
    TRISB = 0x03; //AN0 and AN1 are analog inputs
    TRISA = 0;
    TRISD = 0;
    TRISE = 0;
    TRISF = 0;

    /***************************************************************************/
*    Configure the ADC Module  
***************************************************************************/
    ADCONbits.ADSIDL = 0; // Operate in Idle Mode
    ADCONbits.FORM = 0;  // Output in Integer Format
    ADCONbits.EIE = 0;  // Disable early interrupts
    ADCONbits.ORDER = 0; // Sample the even channel first
    ADCONbits.SEQSAMP = 1; // Sequential Sampling Enabled
ADCONbits.ADCS = 5;       // Clock Divider is set up for Fadc/14
ADPCFG = 0xFFFF;          // AN0 and AN1 are analog inputs
ADSTAT = 1;               // Clear the ADSTAT register
ADCP0bits.TRGSR0 = 0xC;   // ADC module will trigger on a tmr1 period match
ADCP0bits.IRQENO = 1;     // Enable the ADC interrupt

/*******************
* Configure Timer1 *
*******************/
T1CON = 0x18;             // Use a prescale value of 0
TMR1 = 0;                 // Reset the timer counter
PR1 = TIMER_PERIOD;       // Set the period register to the specified period of timer

// Set up the Interrupts
/****************************
* Configure ADC Interrupts *
****************************/
IFS0bits.ADIF = 0;        // Clear AD Interrupt Flag
IFS0bits.T1IF = 0;        // Clear Timer1 Interrupt Flag
IEC0bits.T1IE = 0;        // Timer Interrupt is not needed
IPC2bits.ADIP = 4;        // Set ADC Interrupt Priority
IEC0bits.ADIE = 1;        // Enable the ADC Interrupt

long c;
int dc_off=0, amp=0, scn=0, cyc_v=0, start=0, stop=5000, num_c=2,
num_h=1;

/****************************
* Restart Bluetooth module *
****************************/
PORTBbits.RB6 = 1;       //initialize CS pin high
PORTEbits.RE2 = 0;       //Reset bluetooth
wait(1000000);
PORTEbits.RE2 = 1;       //Turn on bluetooth
wait(1000000);
wait(1000000);

/****************************
* Configure Bluetooth Module *
****************************/

/*
* Tell Bluetooth not to send verification back to the PIC
*/
U1TXREG='A';
while( U1STAbits.UTXBF);
U1TXREG='T';
while( U1STAbits.UTXBF);
U1TXREG='S';
while( U1STAbits.UTXBF);
U1TXREG='1';
while( U1STAbits.UTXBF);
U1TXREG='0';
while( U1STAbits.UTXBF);
U1TXREG='=';
while( U1STAbits.UTXBF);
U1TXREG='0';
while( U1STAbits.UTXBF);
U1TXREG=0x0D;
while( U1STAbits.UTXBF);

/*******************************
* Reset the Bluetooth Device *
*******************************/
U1TXREG='A';
while( U1STAbits.UTXBF);
U1TXREG='T';
while( U1STAbits.UTXBF);
U1TXREG='Z';
while( U1STAbits.UTXBF);
U1TXREG=0x0D;
while( U1STAbits.UTXBF);
POT(10000); //Initialize digital pot to 10 kohms
DACA(2500); //Initialize analog ground to 2.5 volts

wait(1000000);
wait(9999999);

/***************************************************************************/
/* Configure sleep mode for * 
/* efficient power consumption */
/***************************************************************************/
U1MODEbits.WAKE = 1; //Wakeup upon receiving a UART interrupt
wait(1);
Sleep(); //Enter sleep mode

/***************************************************************************/
/* Run infinite while loop * 
/* and wait for interrupts */
/***************************************************************************/
while(1)
{
  /***************************************************************************/
  /* Adjust digital pot if * 
  /* selected by user */
  ***************************************************************************/
  if(DP_Flag == 1)
  {
    wait(100000);
    pot_val=chartoint(pot_value,4); // Convert received value to integer
    pot_val = pot_val * 1000; // Convert to ohms
    POT(pot_val); // Set the value of the pot
    DP_Flag = 0; // Reset the digital pot flag
    U1MODEbits.WAKE = 1;
    wait(1);
    Sleep(); //enter sleep
  }
  /***************************************************************************/
  /* Perform Chrono Amperometry * 
  /* if selected by user */
  ***************************************************************************/
  if(V_Flag == 1)
  {
    TMR1 = 0; // Clear the Timer counter
    stop=chartoint(end_or_offset,4); //convert ending voltage to integer
start=chartoint(start_or_amp,4);//convert starting voltage to integer

/*********************
* Initialize Serial *
* data transfer     *
***************************/
ULTXREG = 80;
while( U1STAbits.UTXBF);
ULTXREG = 80;
DACA(start);     // Send DAC start voltage

for(c=0;c<4000000;c++)
{
    if(c==2000000)
    {
        T1CONbits.TON = 1; // Enable ADC and Timer
        ADCONbits.ADON = 1;
    }
}
DACA(stop);      // Send DAC end voltage

for(c=0;c<4000000;c++);
while( U1STAbits.UTXBF); // Terminate serial transfer
ULTXREG = 255;
T1CONbits.TON = 0;    // End ADC and Timer
ADCONbits.ADON = 0;
TMR1 = 0;                  // Clear the Timer counter
DACA(2500);               // Reinitialize DAC to analog ground
V_Flag = 0;                // Reset Chrono Amperometry flag
U1MODEbits.WAKE = 1;
wait(1);
Sleep();                   // Enter sleep
}

/******************************
* Perform Cyclic Voltammetry *
* if selected by user         *
*******************************/
if(C_Flag==1)
{
    scn=chartoint(scanrate,4);  //convert scanrate to integer value
    amp=chartoint(start_or_amp,4); //convert amplitude to integer value
    dc_off=chartoint(end_or_offset,4); //convert dc offset to integer value
    num_c = chartoint(num_cycles,2); //convert number of triangle waves to generate
num_h = chartoint(num_hyst,2); //convert number
of cycles to delay before sending data

start = dc_off - (amp/2);        //identify
max and min values for triangle wave
stop = dc_off + (amp/2);

/******************************
* Find Scanrate              *
* from lookup table          *
*******************************/
for(ind = 0; ind<21; ind++)
{
    if(scn==stepl[ind])
    {
        step = step2[ind];
        break;
    }
}

/******************************
* Initialize Serial          *
* data transfer              *
*******************************/
U1TXREG = 80;
while( U1STAbits.UTXBF);
U1TXREG = 80;

DACA(start);
wait(900000); //step to dc offset before
starting cyclic voltammetry

/******************************
* Perform cyclic voltammetry *
* with user specified         *
* number of cycles            *
*******************************/
for(c = 0; c < num_c; c++)
{
    /***************************
    * Only send data after    *
    * number of delay cycles  *
    * has been met            *
    ***************************/
    if(c==num_h)
    {
        ADCONbits.ADON = 1; //
        T1CONbits.TON = 1; //
    }
}

for(cyc_v = start; cyc_v<stop; cyc_v+=1)
{
    DACA(cyc_v);
    wait(step);
}
/***********************
*  Generate Saw Tooth  *
*  if requested        *
***********************/
if(Triangle_Flag == 1)
{
    for(cyc_v = stop; cyc_v>start; cyc_v-=1)
    {
        DACA(cyc_v);
        wait(step);
    }
}

while( U1STAbits.UTXBF); // Terminate serial transfer
ULTXREG = 255;
T1CONbits.TON = 0; // End ADC and Timer
ADCONbits.ADON = 0;
DACA(2500); // Reinitialize DAC to analog ground
TMR1 = 0;                   // Clear the Timer counter
scn=0; // Reset all values to default
amp=0;
dc_off=0;
num_c =0;
num_h =0;

    start = dc_off - (amp/2);
    stop = dc_off + (amp/2);

    Triangle_Flag = 0; // Reset Cyclic Voltammetry flags
    C_Flag = 0;
    U1MODEbits.WAKE = 1;
    wait(1);
    Sleep(); // Enter sleep mode
}

/***************************************************************************/
*  Analog to Digital Interrupt Service Routine  *
*  Purpose: Sample analog inputs and create discrete signals      *
***************************************************************************/
void __attribute__((interrupt, no_auto_psv)) _ADCInterrupt(void)
{
    int channel0Result, channel1Result, tempi;
    char temp;
    channel0Result = ADCBUF0; // Get the conversion result
    channel1Result = ADCBUF1;
/************
* CHANNEL 0 *
************/

/****************************
* Store first 8 bits *
* to send to LabVIEW *
*************************/
tempi = (int)channel0Result >> 2;

if(tempi==255)   // make sure not to send stop char to LabVIEW
temp=254;
else
temp = tempi;
U1TXREG = temp;

/****************************
* Store last 2 bits *
* to send to LabVIEW *
* thus utilizing full *
* 10-bit resolution *
*************************/

tempi = (int)channel0Result & 0x0003;

if(tempi==255)   // make sure not to send stop char to LabVIEW
temp=254;
else
temp = tempi;
U1TXREG = temp;

/****************************
* CHANNEL 1 *
*************************/

/****************************
* Store first 8 bits *
* to send to LabVIEW *
*************************/

tempi = (int)channel1Result >> 2;
if(tempi==255)   // make sure not to send stop char to LabVIEW
temp=254;
else
temp = tempi;
U1TXREG = temp;

/****************************
* Store last 2 bits *
* to send to LabVIEW *
* thus utilizing full *
* 10-bit resolution *
*************************/

tempi = (int)channel1Result & 0x0003;
if(tempi==255) // make sure not to send stop char to LabVIEW
temp=254;
else
  temp = tempi;
U1TXREG = temp;

IFS0bits.ADIF  = 0;   // Clear ADC Interrupt Flag
ADSTATbits.PORDY = 0;  // Clear the ADSTAT bits
}

/******************************
**********
ERROR CHECKING **********
******************************/
if(U1STAbits.OERR==1)
{
  U1STAbits.OERR = 0;
  return;
}
int Rxchar_counter=0;
Rxchar = U1RXREG; // Store character from the receive buffer

/******************************
********
Read in Digital Pot values ********
******************************/
if(Rxchar == 'P')
{
  /**************************************************************
  * Stored desired Potentiometer resistance *
  ***************************************************************/
for(Rxchar_counter=1;Rxchar_counter<4;Rxchar_counter++)
{
  while(U1STAbits.URXDA != 1);
  Rxchar = U1RXREG;
  pot_value[Rxchar_counter]=Rxchar;
}
  /***************************************************************
  * Set the digital pot flag high *
  ***************************************************************
  DP_Flag=1;
}

/******************************
********
Read in Chrono Amperometry values ********
******************************/
else if(Rxchar == 'A')
{
  /***************************************************************
  * Stored desired Start Voltage *
  ***************************************************************
for(Rxchar_counter=0;Rxchar_counter<4;Rxchar_counter++)
{  while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;
    start_or_amp[Rxchar_counter]=Rxchar;
}

/*******************************
*  Stored desired End Voltage *
********************************/
for(Rxchar_counter=0;Rxchar_counter<4;Rxchar_counter++)
{
  while(U1STAbits.URXDA != 1);
  Rxchar = U1RXREG;
  end_or_offset[Rxchar_counter]=Rxchar;
}

/*****************************/
*  Set the Chrono Amperometry flag high *
/*****************************/
V_Flag=1;

/*****************************/
*  Read in Cyclic Voltammetry values *
/*****************************/
else if(Rxchar == 'C' || Rxchar == 'T')
{
  if(Rxchar == 'T')
  {
    /***************************/
    *  Set the Saw Tooth flag high *
    ***************************/
    Triangle_Flag = 1;
  }
  /***************************/
  *  Stored desired Amplitude value *
  ***************************/
  for(Rxchar_counter=0;Rxchar_counter<4;Rxchar_counter++)
  {
    while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;
    start_or_amp[Rxchar_counter]=Rxchar;
  }
  /***************************/
  *  Stored desired DC Offset value *
  ***************************/
  for(Rxchar_counter=0;Rxchar_counter<4;Rxchar_counter++)
  {
    while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;
    end_or_offset[Rxchar_counter]=Rxchar;
  }
  /***************************/
  *  Stored desired Scanrate value *
  ***************************/
  for(Rxchar_counter=0;Rxchar_counter<4;Rxchar_counter++)
  {
    while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;
    scanrate[Rxchar_counter]=Rxchar;
}
for(Rxchar_counter=0;Rxchar_counter<2;Rxchar_counter++)
{
    while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;
    num_hyst[Rxchar_counter]=Rxchar;
}

for(Rxchar_counter=0;Rxchar_counter<2;Rxchar_counter++)
{
    while(U1STAbits.URXDA != 1);
    Rxchar = U1RXREG;
    num_cycles[Rxchar_counter]=Rxchar;
}

C_Flag=1;
IFS0bits.U1RXIF = 0;  //reset the interrupt flag

void __attribute__ ((interrupt, no_auto_psv)) _U1TXInterrupt(void)
{
    IFS0bits.U1TXIF = 0;  //reset the interrupt flag
}


