SENSOR ACQUISITION AND CONTROL SYSTEM

By

____

Matthew Duncan

____

Jeremiah D. Jinno

____

John T. Ramsden

Date: 25 April 2007

Major Qualifying Project Report submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Approved:

____

Professor Reinhold Ludwig

____

Professor Sergey Makarov

Abstract

The SACS (Sensor Acquisition and Control System) is a hardware and software package developed as a system to be used in conjunction with an aluminum melt detection sensor signal acquisition previously developed by a team of WPI students. This project builds on a preexisting C code firmware, and is used by a graphical user interface developed using Visual Basic.NET. Through this project, the groundwork has been laid for further thermal and electrical conductivity evaluations of molten aluminum. The SACS is a prototype, in line, non-destructive configuration, that allows process control for aluminum sand casting.

1 - Introduction

1.1 - The Current Need

In the industrial market, no online monitoring exists for high-volume aluminum casting. Currently, if post production Quality Assurance (QA) testing shows that there is a flaw in one unit of a batch, then the entire batch is scrapped. This leads to a loss of money and time, which could jeopardize a company's competitive edge in manufacturing markets. If an online system were to be created, flaws in the process would be detected and corrected before there is a need to scrap an entire batch of product, correcting this apparent problem.

Current online non-destructive tests (NDT), including surface inspection, fail to identify internal problems. This defective part is inspected more thoroughly, and scrapped only when many more resources have been expended, after the part has left the assembly line. Many of these defects are caused by the cast filling process, and can be eliminated if an engineer had the ability to look beyond the cast exterior, and see the molten aluminum fill the mold.

1.2 - Past Solutions

Over the years, many types of NDTs have been developed to increase quality assurance of products. These tests vary from simplistic leak (gas) and liquid penetrate testing, to more complex ultrasonic, acoustic, radiographic, electromagnetic, infrared and thermal testing.

However, each NDT method has benefits and drawbacks. In general, as the NDT method becomes more precise, it requires more sophisticated equipment and a large amount of training, each measure adding a large cost factor to the production process. In the past, all tests were pre and post process qualification tests, where only a small percentage of the units were tested due to time constraints and additional test-associated costs.

There are two direct results of sample testing. There is the potential risk that a sample unit passes QA testing while other units in the batch are of lesser quality and eventually reach the market. Additionally if a unit fails QA testing, an entire batch of potentially good products are scrapped. Both results lead to decreased production efficiency and a loss in overall profits. Presently, with no added process control, the nature of such NDTs lead to an average higher quality, higher cost, product in the market.

1.3 - Possible Solution

There has recently been significant research and development into the use of electromagnetic sensors capable of being inserted into the production line. Several WPI students have been spearheading this research in an effort to establish a more feasible NDT system. This has lead to the development of a sensor that is capable of detecting the presence of molten aluminum as it fills a sand cast. To date, these sensors have been proven effective at short range, in detecting molten aluminum amidst a high-interference, industrial environment.

This type of NDT would ideally be used in a closed loop system to sense and control the flow of molten aluminum, and improve the overall quality of sand cast products. Our MQP focuses the data acquisition and control signal output capabilities of a closed loop design while also providing a graphical user interface to control the system.

2 - Background

2.0.1 - The Aluminum Casting Process

It is important to discuss the aluminum casting process because the inherent nature of filling a cast with aluminum is where a vast number of the problems arise. This process involves the creation of a mold and also the pouring technique used to fill the mold.

2.0.2 - The Mold

Sand cast molding is considered a simple and inexpensive way to form aluminum precision parts at high production rates. In fact, the aluminum sand casting process is considered to be one of the most versatile methods for casting. Although slower than other casting methods, the molds that are produced via sand casting have the added benefit of being able to be used for the production of larger and more complex designs. Because of this, the sand casting process is considered to be ideal for the engine block casting industry [3e].

The specific type of sand cast used to create the test mold is known as a dry-sand cast. A dry sand cast uses chemical binders in order to keep the sand particles together while creating a form. This is the preferred method of creating a sand cast mold for mass production because it requires no time between creating the mold and filling it.

The majority of the sand used in sand casting today is silica sand, which has many properties that makes it ideal for casting aluminum parts. Perhaps the most important quality of silica sand is that it has a melt fusion point of 1760°C. This is very suitable for casting molten aluminum that has a melt point of around 660°C. Silica sand is also of a very fine crystalline structure. This allows for the construction of very highly refined sand casts with a moderately fine finish [8].

The other method of mold creation is green-sand casting, which requires water to form the mold. If molten aluminum hits the mold while it is still wet, the water will instantly turn into steam and expand rapidly causing an explosion. To prevent a steam explosion it is necessary to allow the water to evaporate. This unfortunately adds more time to the casting process, often causing it to be the more undesirable of the two mold creation methods [4].

The process of creating the mold first requires the carving and tooling of wooden patterns. To create a mold without a pattern inside, the wooden patterns are pressed into two halves of a mold; the upper half called the cope and the lower half, the drag [7]. Example of these patterns can be seen in Figure 1.

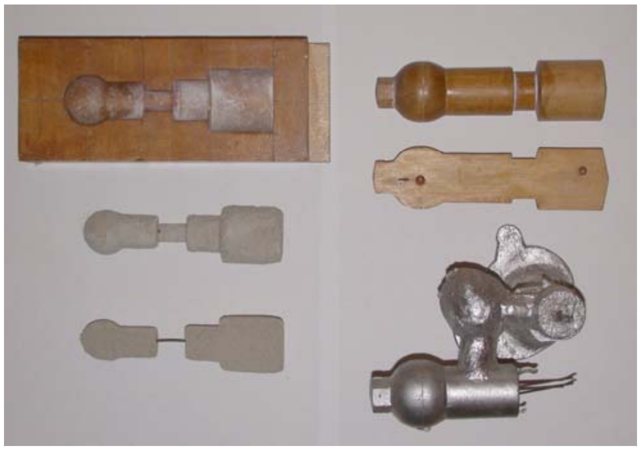


Figure 1: Examples of wooden patterns and cores.

These halves are pressed together to make a negative of the part in the mold, leaving a void which would later be filled with molten aluminum and become the actual cast part.

To create hollow structures within these cast parts, sand cores are positioned inside the mold. When the mold is filled, aluminum flows around these cores, leaving a hollowed-out space inside the part. Only the complexity of the final cast part determines if a single pattern will be used, or if it is necessary to use up to thousands of patterns in conjunction with multiple sand cores.

To ensure correct flow of molten aluminum, a gating system of channels and wells are also imprinted into the sand. These are specifically designed in an attempt to equally distribute the aluminum across mold cavity, and when the molten aluminum cools, the additional aluminum left on the part by the channels and wells can be later machined off [10].

2.1 - Pouring Molten Aluminum

Pouring the aluminum into the cast is perhaps the most crucial point in creating a quality part. If great care is not taken during this stage the mold must be scrapped with a loss of material and time. The reason great care must be taken is due to the material properties of molten aluminum, as discussed below.

2.1.1 - Properties of Aluminum

Pure aluminum can be alloyed with other metals to create a material that is easily formed, cast, or machined. It also has a high thermal conductivity, maintains excellent corrosion resistance and is also extremely lightweight when compared to the leading engine block material, steel. On average, while still maintaining equal strength, a typical aluminum part is 50 percent lighter than its steel duplicate; although it has 50 percent more volume. However, even at the same weight, an aluminum part outperforms a steel part with the ability to absorb twice the amount of energy [3d].

Aluminum will only retain its strength if it is free from any unwanted defects such as inclusions of different materials, voids, or air pockets created in the casting process. During the melting process nitrogen and argon are pumped through the melt to bring all impurities to the surface of the crucible for removal. Once the impurities have been removed from the crucible the remaining molten aluminum is still susceptible to the formation of aluminum oxide (Al2O3) platelets. Aluminum oxide will form whenever air touches the molten aluminum acting as a protective skin for the melt against any other impurities.

Unfortunately this protective skin has the potential to break, causing unwanted inclusions from the mixing of the pure molten aluminum and the aluminum oxide platelets. This dramatically reduces the strength of the part being cast, as the aluminum no longer has a uniform molecular structure. The potential for the protective skin to break is dramaticaly increased during pouring.

2.1.2 - Pouring Methods

There are two methods of pouring that are used with sand casting: A - gravity pouring and B - bottom feeding.

Gravity Pouring takes little equipment to implement and is a favored method for any product where strength is not the primary concern. In this method, the molten aluminum is simply poured from the crucible into the mold. Unfortunately this creates turbulence in the melt and, as explained above, will produce structurally fatal oxide platelets.

For casts that are designed for high-strength parts, such as engine blocks, the addition of inclusions like the ones from aluminum oxide cannot be tolerated. In bottom feeding the aluminum melt is usually pumped against gravity from the firing crucible and fed to the bottom of the sand cast. This method causes little to no turbulence in the molten aluminum as it fills the cast, which allows for the aluminum oxide skin, that forms on the top of the melt, to remain intact. The final result, if this process is preformed at the proper speed, is a part that is free of aluminum oxide inclusions.

The so-called Cosworth Method is a type of bottom feeding that was developed by John Campbell [12]. It is generally used in very high performance part casting. It employs the use of an electromagnetic pump that can supply molten aluminum directly from the middle of the molt reservoir to the cast. The molten aluminum that is taken from the center of reservoir is typically the best to use in casting because it is free from any impurities. Another important advantage of the Cosworth Method is that, once the mold is filled, it is flipped and left upside down. The gating system in the mold then acts as a riser allowing for sand casts to be made without risers, thus speeding up mold production. This can be seen in figure 2

This last method has increased the performance of aluminum engines dramatically. With the ever increasing sophistication of the molds being filled with this method, there needs to be more control over the pouring process. Currently at engine plants across the country the control of the flow of aluminum is done manually.

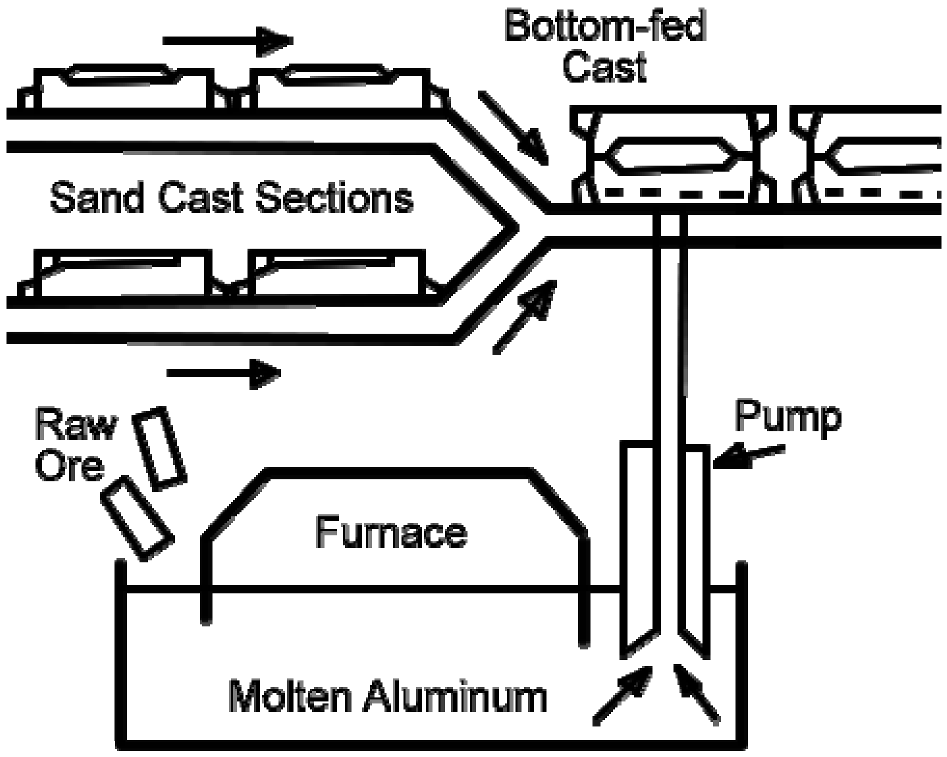


Figure 2: Example of Cosworth Method. The center of the picture the shows filling method.

That is, a worker through time, gains an innate knowledge of how fast or slow the aluminum should flow through the cast. This is by no means a controlled process. The ability to "see" inside a cast in conjunction with a good knowledge of where the aluminum is (and its flow velocity) would prove indispensable to automotive manufacturers.

3 - Objective

3.1 - The Problem

The lack of an on-line testing procedure leaves the aluminum casting industry vulnerable to costly production faults. With the design of an electromagnetic sensor by Brian Foley, Sean Halinan, and Hans Jenson, aluminum can now be effectively monitored from outside the cast without the use of costly and potentially dangerous equipment.

To improve the process of monitoring aluminum casting, the proven capabilities of the sensors need to be utilized in order to effectively control the flow of aluminum. Monitoring and controlling the flow of aluminum in the casting process can directly impact the quality of the final product. Refining this process control could result in a higher efficiency and reduced cost of the entire casting operation.

3.2 - Proposed Solution

The objective of this MQP is to produce a system capable of monitoring several analog sensors, comparing their values against an ideal profile, and providing a system feedback. Based upon the deviation of the received sensor input from an ideal profile, an error signal response could be calculated, displayed, and adjustments could be made to correct the flow rate of molten aluminum.

To measure and compare data, the system must contain adequate memory resources, and be easily configurable. This would allow for the system to hold the ideal profile in memory, which is necessary to create a feedback signal, and would allow for each new profile to be recorded.

Visual feedback needs to be incorporated to actively monitor the error between the current and an ideal fill profile. This was intended to be a meter of lights where green is to be interpreted as being under 5% error, yellow as being between 5% and 10% error, and red as being greater than 10% error. The main feedback however, is an interface to an external pump.

The interface to the external pump was intended to be a Proportional Integral Derivative (PID) control system. In this way, the SACS would be able to control a molten aluminum pump, adjusting the flow of molten aluminum into the cast to keep it within a certain range of the ideal profile. This would allow for added control in the pouring process, decreasing the number of defects found in post production QA testing.

Our system must not only be user-friendly and fully customizable, but must also remain low cost and highly maintainable.

4 - Implementation and Methodology

To achieve the goals of this project, it was necessary to develop both hardware and software solutions that would be fully customizable. This allows for the interface to be continually modified based upon the iterative and incremental nature of our project. Much like any software development project we developed simple working programs first then continually created more functional programs.

4.0.1 - The SACS

The system that was created during the course of this MQP functions very much like a data acquisition and pre-processing card. It has the capability of sensing up to 5 analog signals and recording that data to a computer to be analyzed later. It also has the ability to produce an output signal that is a 5 bit voltage. Uses for this signal will be discussed later in the report.

The SACS can perform these functions at a very low cost, of less than $40. The firmware and computer based software needed to operate the SACS is all open-source, and therefore fully developable. A C-based development package was used to revise the firmware and Visual Basic.NET was used to create the user interface on the PC side of the system. These unique benefits are all possible with the use of the Universal Serial Bus Bit Wacker (UBW), a Programmable Interrupt Controller (PIC) based development board that is the heart of our system.

The UBW itself can handle the reception of analog signals in a laboratory setting at a sampling rate of 100Hz. In order to use the UBW as DAQ in the real world, a buffer was needed to condition the signals before they entered the UBW. Properly shielded cable were also need so that the signal received from the sensors would not be distorted by any stray electro-magnetic (EM) activity that may occur in the industrial setting.

4.1 - The 'Universal Serial Bus Bit Whacker'

4.1.1 - Introduction

The UBW (USB Bit Whacker) is the brainchild of Brian Schmalz. This device is a PCB (printed circuit board) built upon the structure of a USB compatible, Microchip

PIC18, 8-bit microcontroller. The UBW has gone through many revisions and numerous different firmware versions using several of the Microchip USB compatible microcontrollers.

In short, the UBW is a $15 to $20 implementation of a simple parallel/serial data interface. Although this USB interface was originally designed for an educational course at the Science Museum of Minnesota's Learning Technologies, the universal nature of this device allows it to be used in almost any small-scale analog/digital sensor project.

The UBW can be purchased in a 19-contact, PIC18F2455, 3-LED (light emitting diode), 2-button, USB boot-loader programmable form from SparkFun Electronics [10]. This packaged version of the UBW hardware design is an inexpensive and is easily integrated for rapid prototyping. The default firmware that accompanies the processor is designed with a command interpreter for basic input/output control via an RS-232 (standard serial port) over USB connection.

With the use of the RS-232 serial port protocol on a USB medium, the connection and use of the UBW is simplified. The UBW, when connected to a Windows based PC, only needs to have the appropriate Microchip CDC (Communication Device Class) USB drivers installed to work. The UBW will appear as a virtual COM port and can be communicated with using a standard HyperTerminal connection.

One of the primary benefits to the UBW is in the Microchip USB Boot-loader. This enables the device to be re-programmed without needing an external programmer. A secondary benefit, beyond the features of the PIC microcontroller itself, is the low power requirements of the UBW. This allows the entire system to be powered via the USB 5V supply, thereby eliminating the need for a bulky external power source.

For use within our project, the UBW provides an efficient means of analog to digital conversion. The firmware currently supports the automated sampling of ports and transmission of data via USB to a computer in ASCII (standard Latin-based text) format. The sampling signal takes place at a current maximum rate of 100 Hz (the theoretical maximum is 100 kHz) and with 12-bit resolution.

To date, the latest version of the firmware is D v1.4. This firmware and hardware, designed by Brian Schmalz [9], is an open design and can be used and modified for any implementation without permission.

4.1.2 - Hardware Development

4.1.2.1 - Analog Input and Output to a Microprocessor

Analog signals can be anything in the real world that can, or can not, be experienced by humans. This may include light ranging from infrared to ultraviolet, sound waves, subsonic to ultrasonic, pressure, voltages, currents, etc; all of which are continuous signals that change as time progresses.

It is theoretically not feasible to perform calculations in real-time with analog values because computations take time to carry out, and can only be done with discrete numbers. It is also impossible to create an ideal system that can process all possible amplitudes of any analog signal. In fact, if an analog signal is received that is too far out of the range of the measuring instrument, it could very easily cause damage to the processing system. For example, loud music damages ear tissue and audio recording equipment, and ultraviolet light can cause damage to eyes and photography equipment. For this reason it is essential to protect the I/O (input and output) of the system that is subjected to an industrial environment, and limit the analog input signal to one that only includes the range of interest for more precision.

In the case of the SACS (Sensor Acquisition and Control System), we observe the voltage signal which represents the level of aluminum affecting the eddy current sensors. This voltage is an analog signal which varies continuously in time and needs to be turned into a digital 10 bit number by the A/D (analog to digital) converter imbedded in the PIC18F2455 of the UBW.

With this sample of a real world signal it is possible to perform calculations, and create a signal which holds the relevant information. Using this output, a computer system can be put in place to interpret the resulting data and make decisions based upon an underlying algorithmic framework.

4.1.2.2 - Circuit Protection

Voltage Clamp

The output range of the sensor is between zero and five volts, which is within the safe limits of the PIC18F2455 A/D converter. However, there is a significant chance that a voltage outside the safe range can be introduced through a malfunction in the sensor, human error, ESD (electro static discharge), or through EM noise. If the signal voltage is outside the bounds of this safe range, for instance from negative 1V to positive 6V, damage could occur. The solution is rather simple because we have a constant ground and a five volt supply from either the USB supply, or an integrated supply. A voltage clamp circuit implements the characteristics of a diode to keep the input from going too far above one rail or to far below another rail. The design can be seen in Appendix B entitled Voltage Clamp.

This module has an input, from the sensor, an output to the PIC18F2455 A/D converter, a +5V rail, and a ground rail. A diode is placed between the input and the 5V rail so that when the input voltage exceeds 5.7V, the diode turns “on”, developing a voltage drop of 0.7V causing the output to not exceed 5.7v. Another diode must be placed between-ground and the signal rail to limit the signal from exceeding -0.7V. This functions exactly in the same way as the other diode. A resistor must be placed after the input signal, before the diodes to dissipate the power in the case where the input is outside of the -0.7 to 5.7 voltage range. Without this resistor the current would not be limited, and the power would be dissipated by the diodes causing them to fail. This system was tested and successfully limits the output of this module, to a safe range for the A/D converter of the PIC18F2455.

Filtering and Shielding

In an industrial environment with inductance furnaces there are constantly changing powerful EM (electromagnetic) fields, which have the potential to damage equipment, and induce noise on signal wires. Adding noise to a signal can lead to false data points, faulty calculations, and in a control system could lead to adding improper feedback into the system. Induced noise is the reason why it is essential to properly shield and filter the SACS. To limit the amount of EM noise that enters the SACS, a layer of metal should surround the internal components and be tied to ground. Another source of noise emanates from the connection cables. The SACS uses coaxial cables, which are innately shielded, to interface with the sensors and a shielded USB cable which interfaces with a PC. However this shielding cannot eliminate all possible noise sources. Thus, a low pass filter must be inserted along the signal line to remove any additional noise. Since the signal line is sampled at a maximum rate of 100 samples per second, any changes in the voltage that are faster are not of interest, and could cause problems. For this reason a simple RC filter was put in place with a cutoff frequency of 150 Hz just above the sampling rate.

Buffering

The SACS uses unity gain buffering to ensure that the recorded voltage accurately represents the output of the sensors. If the SACS were to draw too much current from the sensor, the resistance of the cables, as well as the voltage clamp, would cause a voltage drop. The buffering module allows for an extremely large input resistance, which limits the flow of current, and ensures a low output resistance. The buffer module also has the same voltage at both the output and input, thereby achieving the goal of not loosing precision.

4.1.2.3 – Unicersal Serial Bus Compatibility

Universal Serial Bus (USB) was developed based on three major considerations: communication, ease of use, and port expansion. The developers have strived to create a standard to transfer data through inexpensive available hardware and in a way that outperformed many other types of interfaces. This fast inexpensive accessory allows for the use of plug and play peripherals without the need of installing additional adapter cards, or to work with Derivative Integral Proportional (DIP) switches or Interrupt Request (IRQ) settings.

These goals were achieved by the USB2.0 interface. Today every personal computer has multiple USB 2.0 ports, making them the prime candidates for any project that requires interfacing with a computer. The usage of USB enables the operator to quickly communicate and program with ease. USB2.0 also allows for three different types of data transfer including 1.5Mb/s, 12Mb/s and 480Mb/s. These data rates enable the interface to run a program on a PC which sends commands over the USB interface well over 100 times a second. The USB standard holds provision for a supply of 5V, 0.5A, which for many small applications is sufficient. The USB interface is therefore superior to other interface types that often are slower, need an expansion card to be installed into a PC, or do not have sufficient power to supply a peripheral.

4.1.2.4 - PIC Microprocessors

The benefit of using a PIC microprocessor is primarily development speed. Free sample microprocessors are available from the Microchip Company for a majority of their products.

PIC18 Family of Microprocessors

Microchip, in its development of the PIC18 family of microprocessors, has produced an ideal solution for rapid-prototyping of small-scale ADC (Analog to Digital Conversion) devices. With a minimum of 10-bit resolution on its analog inputs, this 8-bit microprocessor family can sample data at a maximum rate of between 30 and 200 kS/s (kilo-samples per second). Several of the microprocessors are inherently USB 2.0 compatible, and can run internal firmware at speeds beyond 48MHz.

The Microchip C18 Compiler & Boot-loader

The benefit to using the PIC18 family of microprocessors extends beyond their data-acquisition and data-processing capabilities. In particular, Microchip provides a well-documented C compiler. Moreover it provides self-programming (over USB) Boot-loader firmware which eliminates the need for an external microprocessor programmer.

C18 Compiler

The C18 Compiler is one of the various compilers available from Microchip. It is available to anyone who registers on the Microchip website for a 60-day free trial. After the 60-day trial, the feature-set becomes limited, but will still allow use. The compiler license is available for purchase through Microchip.

The C18 compiler appears superior to other C compilers because of the depth of documentation. Beyond this, the C18 compiler works directly with Microchip's MPLab with minimal configuration, and allows for simple C code syntax to be used instead of the traditionally accepted PIC assembly language constructs.

USB Boot-loader

A firmware boot-loader and computer application is provided by Microchip for use with its USB-compatible microprocessors. This allows for the microprocessor to be programmed and re-programmed while remaining integrated within the application circuit. It is accomplished by providing a serial means of programming to the microprocessor, and is transferred using a standard USB 1.0/2.0 connection from a computer.

The default provision of the boot-loader allows for the microprocessor to enter "programming mode" upon restart. The means for such a switch is programmable; it allows for an application specific implementation. Most commonly this switch is implemented as a hardware jumper that is held at low potential. This is enables the microprocessor to be reset.

4.1.3 - Firmware Development

At the date of the beginning of this project, firmware D v1.3 was the latest released firmware. There were several modifications made to this firmware, to better facilitate use within our project.

4.1.3.1 - Enhanced Case Sensitivity

The UBW originally comes with the ability to send ASCII commands via a HyperTerminal Connection. Unfortunately, this is limited to upper-case letters, making accidental key-stroke errors more common. Since there is a rather limited number of commands, there is no need to have both upper and lower-cased letters identify different commands.

The first modification made to the UBW firmware was the added support for case insensitivity. This allowed for commands to be sent without worrying about the case, and directly increased the ease-of-use of the device. This enhancement was madein an effort to make automated interfacing of the device simpler for both human and application interface.

4.1.3.2 - Modified Return Statements

Each of the commands sent to the UBW respond with a notification of success or failure. The UBW comes with a single default response for successful command execution, and comes with numerous responses for various different command-entry errors. With a lack of command-specific responses, an automated interface to the UBW would be unable to identify if a command had resulted in a correct response from an alternate command.

The modification was to add command-specific responses to be sent back upon successful execution of the command. As a result, several additional limitations and/or inefficiencies were discovered, one being the inconsistent return method for ASCII text.

4.1.3.3 - Enhanced ASCII Return Efficiency

All of the UBW commands respond by sending ASCII text, character by character, over the USB interface. This is achieved through use of a USB buffer in memory, which is wiped with successful transmission of a character, and loaded with the next character to be sent. There are, however, different ways of loading this buffer. One entitled "putUSBUSART()" requires that the characters be in a globally accessible array of characters. Another, "putsUSBUSART()", takes a string literal as an argument, and loads the USB buffer one character at a time until the string has been exhausted. This last function can be called by passing a new string directly, or by passing a pre-defined string variable. The inefficiency of the original UBW firmware lies in the assembly-level construction of the code.

Although the firmware was written in C, it is cross-compiled into PIC microcontroller assembly before it is compiled into a hexadecimal-encoded file to be programmed onto the device. This means that, although the code appears to be C, it is actually making simple assembly substitutions for the C operations. For the UBW, this implies that every time a string literal was parsed and passed as a new string, more memory is being used to store the same response in multiple places.

To reduce the memory overhead for future use/advancement of this device, all reoccurring string values are turned into read only memory (ROM) variables. This makes it possible for any function to call the string from a single location within memory rather than re-allocating additional memory for each additional call.

4.1.3.4 - Removed Hard-coded LED application

The original UBW firmware has the two accessible LEDs hard-coded into monitoring and displaying USB connectivity. Although this is an excellent feature for initially testing the functionality of the device, it hinders testing of situational output based upon an input signal.

The removal of this USB connectivity monitoring in no way alters the functionality of the device. It simply provides access to both LEDs on the UBWs port C.

4.1.3.5 - Added Communications via SPI

The preferred method of interfacing with the external DOSonCHIP SD (Secure Digital) memory card module is via Serial Peripheral Interface or SPI. This is because SPI supports full-duplex (bi-directional) communication, high throughput, and needs no addressing, reducing overhead in device access. Because of the use of SPI, several additional internal functions were added for use with the DOSonCHIP module. This was implemented using the Microchip C18 SPI module.

The SPI functions are not completely implemented when evidence was found that the DOSonCHIP was not capable of providing the transfer rates desired for our project. However, the theoretical implementation is described below. This implementation is based upon a sample-code implementation for use with a PIC30F4011. The original code can be found in Appendix C.

NOTE: These command structures were later deprecated due to the huge decrease in performance suffered through use of the DOSonCHIP module. (See section "Implementation and Methodology: Pitfalls – External Storage using Secure Digital Media")

Setup Ports for SPI Communication

To use the ports on a PIC microcontroller for data transmission to the DOSonCHIP module, the data direction needs to be set correctly on each tri-state port. Following this, the SD module's reset and chip select pins are held low, forcing an acceptance of the new configuration.

Additional functionality is provided by the sample code in Appendix C, allowing the SPI setup to be functional through modular code. This means that if it were necessary to add additional functionality, the entire code would not need to be modified, but merely a lower-level function.

Initialization of UART

This is done in two functions because of the initialization process required to use SPI. First and foremost, the DOSonCHIP, in conjunction with the SPI communication standard, requires that the frequency during initialization be limited to 8MHz. After initialization, the frequency can be increased, but without proper initialization further communication is not possible.

Sending Data to the SD Card

A nano-scale delay is generated between setting the Chip Select and writing to the buffer. This is to allow the DOSonCHIP to accept the incoming data. From here a check is done to see if the SD card module has something to communicate in response to the data sent. If it does, the routine allows for the data to be sent by providing a micro-scale delay for the response to be received.

4.1.3.6 - Added Unique Identifier for Versioning

One of the original UBW features was a string variable that contained a text copy of the version of the UBW. This allowed for a simple version-checking command to be sent to the UBW receiving the version back as the response.

The enhancement is to provide a unique identifier for this particular UBW implementation. To do so, the version string literal was changed to “UBW.D.130”. Although this could have theoretically been anything, the concept with this formatting change was to allow for a more universal, and expandable, versioning format. This format is designed to be more easily parsed and identified by a computer application, while retaining the pertinent visual information.

4.1.4 - Software Development

Utilizing the functions of the UBW via a Hyper-Terminal is simple and straight forward; however it is not automated, and does not allow for manipulation of data as it is being saved. To add these capabilities to the user interface of the UBW, the command-line interface was dismissed in favor of a graphical user interface.

The GUI (graphical user interface) allows for multiple commands to be automated in both interactive and non-interactive methods. This implies that not only can a user execute a series of commands with a single interaction, but the application itself can algorithmically execute commands for certain purposes as well. It is this fact that proves to be the most valuable asset to our system.

There are many different programming languages, and a large subset of these allow for GUI development. Unfortunately much of the programming resources on creating an RS-232 (serial port) interface (virtual com port or not) are very expensive. In order to combine these two features, and to stay within the project budget, an open-source solution in Visual C/C++ is ideal.

Unfortunately is extremely difficult to find any open-sourced software that successfully managed to implement a virtual com port connection. The only viable option was an entire open-source terminal application that could be analyzed and re-invented to support our desired automation.

Specifically we saw a need to automate several processes. First, the ability to find what port the UBW was located on and then successfully running the appropriate configuration could greatly increase the usability of our device. Finally, the most important automation desired was the ability to automate the collection and storage of data. Because of these basic needs, and the lack of available software implementations in C/C++ (however ideal), forced us to adopt Visual Basic .NET as the language of choice.

4.1.5 - Graphical User Interface Development:

The goal in our graphical interface was to create a program that allows the user to control and collect data without leaving the confines of a single interface. Our interface also is designed purposely to reduce configuration overhead and reduce human error. By developing our interface from Microsoft’s common form elements, we can avoid violation of user expectations.

4.1.5.1 - The UBW VB.NET Class

Due to the nature of the device, to directly interface between the Visual Basic .NET application and the UBW is not ideal. A layer of abstraction is desired both for the RS-232 communication functionality and for the UBW command functionality. Given that Visual Basic .NET is an OO (object oriented) programming language, it was an ideal solution to design this needed abstraction as a Class structure.

Interestingly the RS-232 class module was already available, as freeware, but the UBW needed to have an entire OO class structure built from the ground up. To insure that all needed functionality is carried through to the end user, the UBW class was created simply as a direct parallel to the UBWs ASCII-based command structure.

The UBW class consists of a single parallel function for each available function on the UBW. It also contains the ability to construct a UBW object using either a new or pre-existing RS-232 connection. This allows for a single UBW connection session to be passed between different functional sections of the application. Finally, the UBW contains both a direct-connect and search-connect method of opening its own RS-232 connection as well as getters and setters for all associated private member variables.

Although there are many methods inside the UBW class, the majority of them are left untouched for the purposes of this project. The entire method list and explanations on the functionality of each can be found in Appendix D. Only methods used in the SACS project are described below.

4.1.5.2 - UBW Class Properties

Port Question

The UBW class contains a property called PortQuestion() that can be used to get or set the unique identifier string. By allowing this, it is possible to have numerous different SACS devices attached to a single computer, each potentially recording values from different input devices.

The default identification string is set to match the default firmware version. This string currently is “UBW.D.130”. The computer will only make a connection to the UBW if the port-question matches the unique id (currently the version string) of the corresponding UBW device.

Connection Verification

The UBW class contains a property called conxd() that returns a Boolean value used to identify the status of the current UBW connection. If the device currently has an active RS-232 connection, this property will display true.

This property is read-only because the function of connecting/disconnecting is left up to a different function.

4.1.5.3 - UBW Class Functions

Private Function: Find COM Port

This function, findComPort(), is designed to systematically search through 100 COM ports/Virtual (VCOM) ports. It will return an integer port number upon successful verification of the UBW's unique identification.

The location of the UBW is determined by looping through a large number of COM & VCOM ports. On each port, a connection attempt is made to access the UBW. Failure to connect determines the absence of a UBW. Connection, however, does not determine whether or not to verify a UBW existence on a particular port.

Because there is the possibility to connect multiple UBW devices to the same computer system, the port question (See "*Software Development - UBW Class Properties: Port Question*") is used to verify that the UBW currently attached to the computer system is the correct one for the desired system performance.

When the UBW firmware version is verified against the port question, the port number is recorded, and returned from the function.

If there are multiple UBW devices connected with identical firmware names, they will register on two different VCOM ports. For the simplification of the project, only the first of identical UBW devices will be connected to. However, once a connection is made to a port, another connection attempt to the same port will result in a failure. This would cause the findComPort() to return a '-1' while looking to make a new connection to a UBW with identical specifications and versioning.

Public Function: Connect

This function is actually the abstract interface to the findComPort() function, allowing a connection to be made in a single function call, given setup of or defaulted connection properties.

Public Function: Serial Open

This function and its overloaded version allow for a serial port connection to be made. For the most part, this connection would be used for internal UBW class functionality. However it does allow for access to the serial connection directly from the top level.

The first version of this function operates simply by using the default UBW class values for COM port, baud rate, bit stream length, parity bits, number of stop bits, and buffer size. The overloaded version takes in an integer value for connecting to a specific COM port. Both functions return Boolean true upon successful connection and Boolean false otherwise.

Public Function: Disconnect

This function is a direct abstraction of the RS-232 connection's close() function. This function has the added feature of changing the UBW class's "connected" Boolean to false. This function returns true if the disconnect is possible and false otherwise.

4.1.5.4 - UBW Class General Utility Methods

ID Response

This method is used to identify what the UBW did in response to any particular command. This method works only because every command method of the UBW class passes the actual response from the UBW firmware through this method.

There are eight command specific success responses, five specific error responses, and a catch-all general error response. The purpose of this method is to translate the otherwise cryptic response of the UBW hardware into a more meaningful response for the interface user.

4.1.5.5 - UBW Class Command Methods

Configure

This is used to configure port a, b and c pin directions and also to decide how many analog pins to initialize. Analog pin initialization is explained more accurately in the appendix as part of the firmware documentation of the configure command.

The basic functionality is simply to pass the configure command to the UBW hardware and await a response. This requires that an 8-bit binary mask be created for each port on the UBW, a 0-set bit indicating an output and a 1-set bit indicating an input. This number then is converted to a decimal number and passed as the port a, b, and c pin directions.

The analog pin initialization is complex to understand, but basically consists of an 8-bit binary number where a pin is specified as analog by being set to a 1. The stipulation to this is that all pins under the highest pin selected as analog must also be analog. Not adhering to this rule will simply cause the analog equivalent of the digital value of the pin's current output voltage to be presented as the only value on that pin.

This could be used as a desired effect, but is otherwise rather useless, and is simply a limitation to the current firmware structure of the UBW.

Version

The version command is used to send back the firmware coded version string. This string is beneficial in identifying application specific UBW devices, and is used in the port question method for identification of the UBW's COM port.

This method requires no input parameters, and returns a string containing the firmware coded version.

Reset

Resets all pins and ports to default values. This eliminates any previous configuration that was operating on the UBW.

This method requires no input and provides no output beyond the command response to identify successful command execution.

Timed Sample

This command allows for a single call to the UBW device to return streaming samples from the analog ports. This runs equivalent to the Analog Sample command on the UBW, but returns based upon a millisecond resolution and an I/O direction. The resolution is currently limited to one sample every ten milliseconds, or equivalently 100 samples per second.

On the UBW firmware side, the benefit of this method of analog input reading is that it eliminates successive calls to the UBW, effectively reducing USB bandwidth usage by half.

As with the Analog Sample command, these values are returned as a string in which each analog port is represented by a comma-delimited, decimal equivalent, 10-bit number.

4.1.5.6 - Connect/Disconnect

The process of connecting and disconnecting via the GUI is made simple through a connect/disconnect button. This button acts as a switch, allowing a currently inactive connection to be switched active, as well as allowing a currently active connection to be switched inactive.

The connection process is an automated series of events in which the UBW class searches for the UBW, locates the COM port, activates a connection, and sets up the UBW with a default configuration. It is important to note that the GUI configuration of the UBW is only enabled when there is an active connection to a UBW device.

The disconnection process is also an automated series of events in which the UBW class resets all current configurations, and sends a termination-of-communication signal via the UBW disconnect method.

Please note that any failure to make a connection will result in an error message box being displayed. Although the message “UBW_000:UBW connection impossible” suggests that a connection cannot be made, it is possible that an active connection to the UBW already exists erroneously, and is producing this error. In such cases, it is suggested that the UBW device be manually reset using the hardware reset button, or by disconnecting and re-connecting the USB cable to the computer.

4.1.5.7 - UBW Port Configuration

Once the computer has established a connection to the UBW, it sends a configuration command. By default the GUI configures the UBW to read 3 analog pins from port A of the UBW. This number can be changed by using the slider provided in the GUI. Please note though that all changes must be made effective in order for the test to be started.

The configuration is done by sending a port-specific directional mask for each port followed by the number of analog inputs. The port mask is a decimal equivalent 8 bit binary number that has 0’s signifying output pins and 1’s signifying input pins. Each bit represents an available pin on the corresponding port. The default value used by the GUI is 7 because this corresponds to the 8-bit directional mask ‘00000111’ telling the UBW to make the first 3 pins of the port to be inputs.

The fourth input parameter to the configuration function is an integer value representing the number of analog I/O pins. The default number of pins, set by the GUI to be analog inputs in the configuration process, is 3.

4.1.5.8 - Automated Analog Sampling

There are five functional parts to the Automated Analog Sampling done by the UBW class: a sampling command, a data recording function, a timer, a reset operation, and a reconfiguration.

Once the UBW has been configured for analog sampling the UBW firmware command for analog sampling can be executed. Once running, the UBW continually samples and sends the current analog value on the selected pins. These values are given as a string of comma delimited, 10 bit binary equivalent, decimal numbers, and are parsed and appended to a plain text file. This file is given the extension .CSV (comma separated variables) in order to be easily imported to Microsoft Excel.

As this process is running, a timer is counting to provide a time limit for the test. At the end of this user specified time (ms) the timer issues a reset command to the UBW. This causes the UBW lose its configuration, however it is immediately reconfigured to pre-test status.

4.1.6 - Using the Graphical Interface

4.1.6.1 - Setting Test Parameters

Once we had the basic functionality of being able to connect to the UBW, we needed to the ability to set up a test to record data from the sensors. The first parameter we wanted the user to be able to define is how long of a test is desired. By running a test we ask the UBW to record data for a certain amount of time. The user can input how much time they would like to run their test in milliseconds in the text box. Once one has entered an amount, the set button is pushed. This writes the number that was entered in the text box to the variable “msecs”. This is a parameter that is passed into the run test function. The next parameter that we wanted the user to define was how many sensors they wanted to take data from. Currently our system has the capability of retrieving data from up to 5 sensors. The user can drag a slider to select from taking data from 1 to 5 sensors. It must be noted that as of this revision the user cannot define which sensor they want to retrieve data from. The user can now hit the configure button which will send an string of instruction to the UBW to set up to test.

It can be noted that at any time before the user hits the Run button they can reset any of the aforementioned parameters.

4.1.6.2 - Storing Data

The user is also granted the ability to specify where they want the test data to be stored. This is done using VB implementation of a windows file system. The user can search hierarchy for a directory they want the .csv file to be stored or create a new one.

4.1.6.3 - Running the Test

If the user is satisfied with the test parameters the Run Test button is pressed. This will send the command to the UBW to constantly poll the pins that were selected by the user. The test will run for the allotted time and all data can be viewed in the .csv that is created after testing.

4.2- Commercial Solution

This report has primarily dealt with creating a customized DAQ. This has its advantages, as mentioned before, of being cheaper and more customizable. However ,there are also commercial, or “off-the-shelf”, solutions. One such solution is the NI-6008 DAQ from National Instruments. This is a fully functional DAQ with 8 analog inputs and 2 analog outputs at a price of $150. The full specifications of this device can be found in Appendix F entitled “NI-6008”. This DAQ uses a proprietary software package from NI in order to program the DAQ and create a GUI to operate it. This software is called LabView.

4.2.1- LabView

LabView is a graphical software development suite. It can be used to develop functional GUIs that can process a variety of data. Instead of coding line by line, much of the programs developed in LabView are created in a visual fashion. Building blocks or Virtual Instruments (VI) as they are known in LabView, are placed on a panel by the user. Then the VIs are connected together to create a functioning program.

This commercial solution was included in the overall development of our project, it became an exploratory endeavor to try and produce a viable solution using the NI-6008 and LabView as a commercial Sensor Acquisition and Control System.

A standard “Hello World” program was created in order to test to the connectivity of the NI-6008 to the computer using LabView. This involved the creation of a waveform display window and then connecting it to a DAQ Manager. The DAQ manager is akin to a MS Windows based “Wizard”. This manager auto-detected the NI-6008 and gave the ability to choose data type and sampling rate.

Since this was a basic test, the default settings of Voltage and 1 kHz sampling were chosen respectively. Once this was placed down on the workspace it was connected to a waveform display. Once the LabView program was set up, leads from a DC power supply were connected to the NI-6008. Voltage was then supplied to the DAQ while the LsbView program was running. The output from the power supply and the waveform display were read and were found to be accurate with each other.

Measuring Height with LabView

Once data acquisition from the NI-6008 to the computer had been verified with the “Hello-World” program, work could be done on more functional LabView programs. The next task that was of interest to complete using the NI-6008 and LabView was to detect the height of the aluminum as it fills the mold. This was done be creating thresholds for the voltage data that would trigger LED displays if the melt reaches certain heights. In LabView we had the voltage data coming from the NI-6008 going into a VI that was a simple “greater than” Boolean switch. This switch would output a “True” whenever the input voltage is greater than a set value. The output of this switch would be sent to a virtual LED that turns on when it sees a True. To finalize this program, three switch-to-LED systems were setup on the main workspace and were calibrated at different voltage levels. When the program was run the voltage from the power supply going into the NI-6008 was turned up and down and the LEDs triggered according to what voltage level had been reached. For an actual implementation there would need to be a separate switch and LED combination for each sensor channel tuned to what voltage is produced by the sensor coil as the molten aluminum reaches the center point of the coil. That way an LED or some sort of indicator would trigger as the aluminum reached the mid-point of the corresponding coil.

Measuring Flow Rate with LabView

The flow of aluminum through the cast is another important piece of information that can be collected from the signal produced by the sensor. This can be determined by taking the rate of change over the course of the data being acquired through the DAQ. This was done by taking the input from the DAQ wizard and connecting to a “point by point derivation” VI. This VI takes the derivative point by point along the data stream and outputs the derivative as an integer. We then took this output and did the same thing we did wit the height program. As the derivative output reached a certain level an LED would trigger.

5 - Field Testing

Experimental operation of the SACS in conjunction with the electromagnetic sensors has been done on small and less complex aluminum sand casts. These tests of functionality were conducted at Metal Casting Technologies (MCT) in Milford, NH. In doing so, the system was able to be proven effective, and allowed for multiple stages of revisions to take place.

5.1 - Testing Procedures

The testing procedure can be broken down into four sections: Setup, Calibration, Test, and Verification. From this baseline we were able to systematically approach each test, identify the positive and negative aspects of the test, and incorporate the new findings into the next revision of the system.

Each test consisted of the filling of one pre-made mold. The molds were constructed using a dry sand casting method. The test encapsulated the process followed by two foundry workers as they poured the molten aluminum in the top of the mold. In these tests, to simulate the filling process as a bottom feed setup would, the mold was constructed so that molten aluminum entered the cast from the bottom.

5.1.1 - Setup

Upon arrival at MCT the first item that was done in preparation for testing the SACS was to erect the sensor coil assembly created by the other MQP team. The sensor coils were aligned at different heights according to the height of the cast used for the test. Then the sensor group's computer, power sources, and oscilloscope were set up. Once the system was powered and the LabView based program was set up on their computer, the SACS was connected to the sensor group's drive circuit. The SACS' USB cable was connected to the test bench laptop (specs for Laptop can be found in Appendix F). The SACS GUI was then booted up. In the GUI the SACS group set the file path for data acquisition as well as the number of sensors that will be used for data acquisition.

Apart from the MQP teams, employees of MCT also set up connection to cast-integrated thermocouples that had been specially placed in the sand casts used in these tests. The thermocouple data was to be used as a benchmark for our system. However, due to some difficulties with the DAQ hardware used by the thermocouples no such benchmark could be made.

5.1.2 - Calibration

At the time of testing, the sensor group had yet to implement the automatic re-resonating circuit. This meant that the group had to tune their driver circuit by means of potentiometers in order to bring all of the coils into resonance. This was often the trickiest part of each test because the driver circuit needed to be adjusted every time.

Calibration for the SACS is not possible at this time. All data that is not between the minimum and maximum voltage potentials of the SACS inputs is considered erroneous, and therefore does not cause any re-calibration.

As for software configuration of the SACS prior to testing, 3 inputs were specified, and were given a folder in which to place the data acquired. Primarily the tests were set to record at 30 second lengths. Extended testing was defined as a 1 minute 30 second record length.

5.1.3 - Testing

With the sensors properly aligned and calibrated to the test environment, testing was ready to commence.

The filling of the casts at MCT is done by hand. Therefore the accurate synchronization, of both the pouring and the activation of both MQP systems, was absolutely essential.

Each test fill begins with the employees at MCT preparing a bath of molten aluminum by filling a crucible with melt from the foundry. Then, once carried by crane to an area in front of the test mold, special handler rods were attached to the crucible so that the workers could tilt the crucible once the crane lifted again.

When the crucible was lifted the second time it was ready to be poured. A timer was used to synchronize the pour of aluminum and the acquisition of the data from the sensors. The molten aluminum would be poured at a different rate for each of the three tests. The employees would stop pouring when the aluminum reached the top of the inlet hole.

The results that were collected during the time spent at MCT can be found in the results section of this MQP.

5.1.3 - Verification

Post-test verification was done through opening the comma-delimited data records in Microsoft's Excel. Once open, the recorded digital voltages were converted to their analog values. This was done using the following equation, where DigValue represents the 10-bit sampled value received, AnaValue the analog equivalent voltage, 1024 the maximum possible 10-bit sample value, and 5 the voltage scale from 0V.

$$\left(\frac{DigValue}{1024}\right)*5 = AnaValue \qquad (1)$$

From this point, the data points were normalized within a range of 0-5V. In the acquisition of sensor data, a low-pass filter was desired, but was unavailable at the time of testing. To mimic this effect, all acquired data was passed through a 50-data-point averaging algorithm.

This provided an accurate, voltage-consistent, model of the level of aluminum within the mold. The results from these tests and the normalized graphs of the sensor data can be found in the results section of this report.

5.3 - Safety Considerations

Due to the industrial environment that is present at MCT both MQP Teams needed to special precaution not only protect themselves but the persons around them. Independent of following general OSHA standards we were asked to be mindful of things go on around the foundry such as fork trucks that delivered the sand casts to and from the testing area.

6 - Results of the SACS

6.1 - General Performance

The test results obtained from the SACS trials at MCT proved the viability of the SACS as a data acquisition unit in an industrial setting. The success of the SACS has shown that there is a cheap and reliable alternative to current means of industrial data acquisition using expensive off the shelf DAQs.

Below are the post-processed plots of the data that the SACS collected from the sensors during three individual tests at MCT. The first two tests were conducted with the same physical sensor locations. The third test was set up so that the sensor used on the bottom of the mold was placed perpendicular to the rest of the array.

In each of the plots there are three averaged lines of data points. Each line corresponds to a 50 sample averaged (noise-filtered) signal from one of the sensors.

6.1.1 - Test One

Depicted on the following plot is the output recorded using the SACS imported and normalized using Microsoft Excel. The bottom most line represents the data from the middle sensor coil. The middle line represents the sensor coil that was at the bottom of the mold. Finally the sensor coil that monitored the top of the mold is represented by the top line.

The reason behind the varied voltages between each of the sensors was due to there positioning against the mold and the inherent cross resonation between the coils.

The positioning affects the magnitude of the voltage drop while the coil is sensing the aluminum. This explains why the sensor coil in the middle was affected the most by the aluminum. During testing it was this coil that had its electromagnetic yolk, or vision of the mold, completely crossed by molten aluminum. The other sensors saw only partial coverage of the molten aluminum due to there positioning.

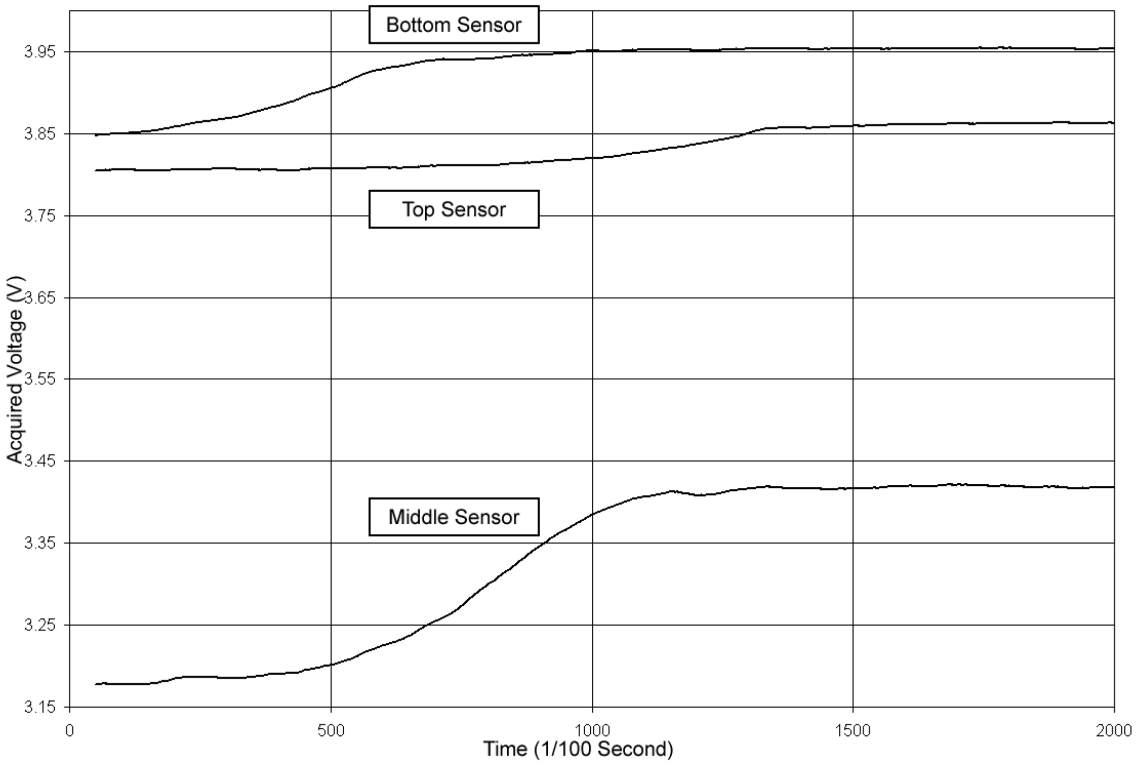


Figure 3: Test 1 - Averaged normalized voltages over 20 seconds.

This first test was an achievement in f itself. Previous testing had be done beforehand in a laboratory setting to make sure that the SACS could actually communicate but there was no absolute way to test that the SACS could actually work in an industrial environment reliably.

It can be seen how the molten aluminum was flowing up the mold by looking at the order in which the sensors detect the aluminum. In the Figure 3 it took approximately 1500 samples to complete the test. Since the SACS system samples at 100Hz this shows that the entire mold was filled within 15 seconds. Near the ending point in the fill (Time > 1500ms) each voltage curve becomes linear.

It was also determined after the test that the USB cable was draped across the BNC connectors we used to connect our device to the sensor coil driver. However, even with these discrepancies it can clearly be seen the SACS collected data that could still show the movement of molten Aluminum through the mold

6.1.2 - Test Two

After correcting the USB interference problem, the second test ran in a similar way as the first test. As expected it can be seen in the plot reported in Figure 4 that the voltage values vary from that of the first test. This was in part due to the fixed interference problem and also partially due to the cross talk that occurred between the sensor coils.

In this plot, the top, middle, and bottom lines now correlate with the data collected from the middle, bottom, and top sensors respectively. The critical points to take note of are not the start and end points, but rather the point at which each line reaches its maximum slope. It is at this point that the aluminum within the cast exactly matches the height of the corresponding sensor.

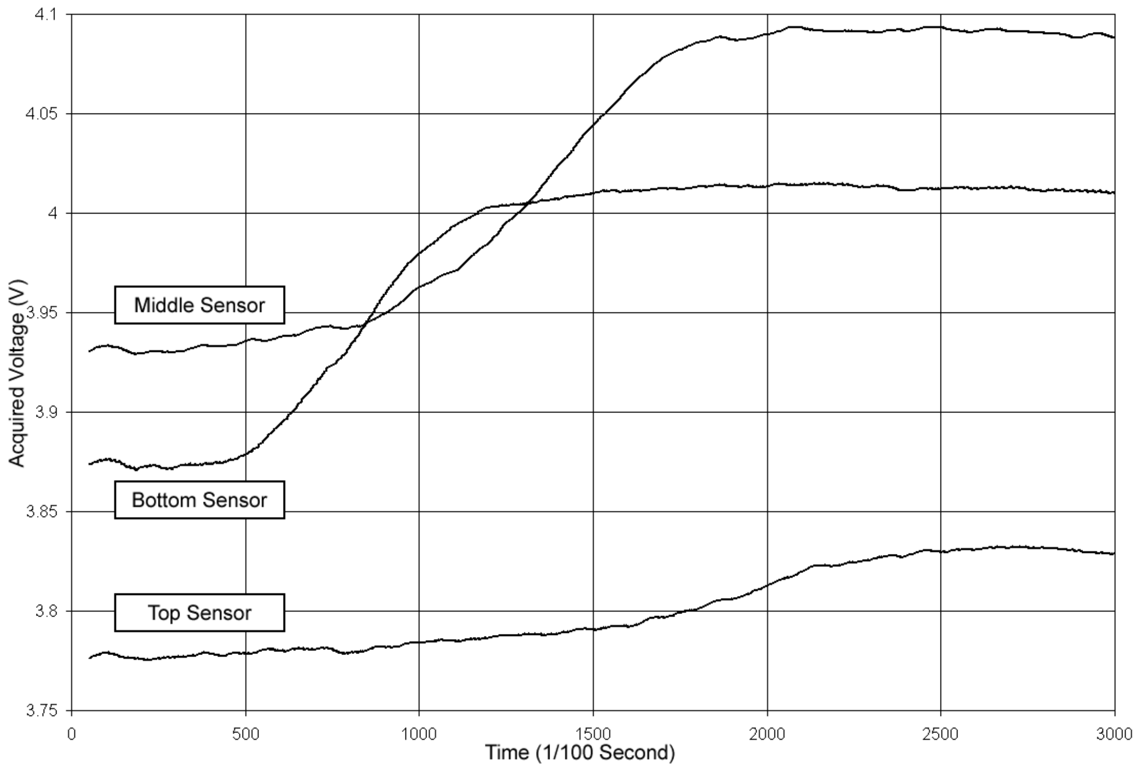


Figure 4: Test 2 - Average normalized voltages over 30 seconds.

The molt in this test was poured at a slightly faster rate. This can be seen in the chart above by the increased slope. This proves the viability of yet another feature of the SACS: that it can collect enough data to derive the fill rate of the mold. Although the derivation of fill-rate is not currently part of the released product, it will be integral to the later implementation of a PID aluminum pump control system.

A second (extended time) plot was actually recorded for this test, whereas only a subset of these points was used to produce the previous plot. This extended plot, see Figure 5 shows a test time of over 100 seconds.

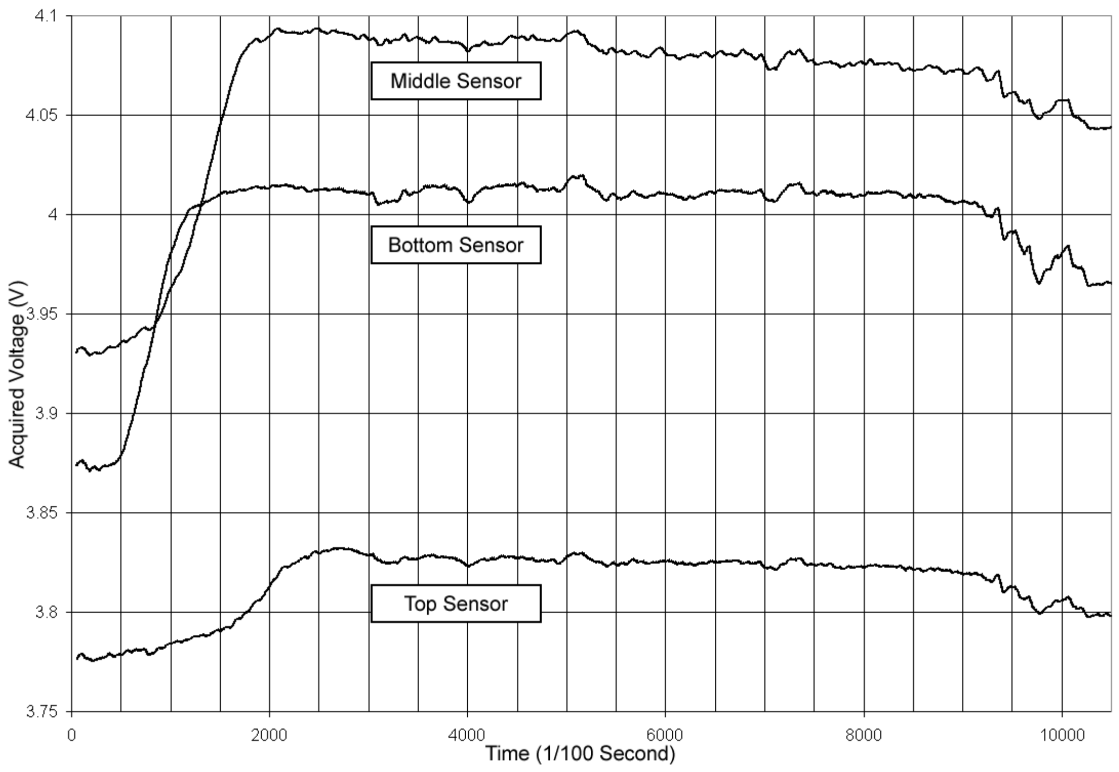


Figure 5: Test 2 - Averaged normalized voltages over 120 seconds.

From the extended plot, there are certain noticeable anomalies beyond the fill-height of the mold. As the molten aluminum begins to cool, its effect on the electromagnetic yoke of the sensor changes, causing a noticeable drop in voltage.

Unfortunately no other extended electromagnetic permeability data on molten aluminum exists to our knowledge that would allow us to prove or disprove the validity of these phenomena. However, looking at the plot, it is clear that whatever produced the voltage drop os almost perfectly consistent across all three sensors simultaneously. These anomalies should be noted, and possibly further developed to identify their nature, meaning and implications.

6.1.3 - Test Three

The third test that was run at MCT had a different coil setup than the previous two tests. The top-most coil, in respect to the mold, was placed on a side perpendicular to the other coils. This was done to see what kind of a response the sensor coil would get when looking through a different width of sand but primarily to see if the perpendicularity of the sensors electromagnetic fields would allow for better detection.

The resulting post-process plot shows that unfortunately there was only a decrease in the response by the sensor that was moved. Although this was expected due to the difference in mold thickness, the desired signal enhancements were not seen at all.

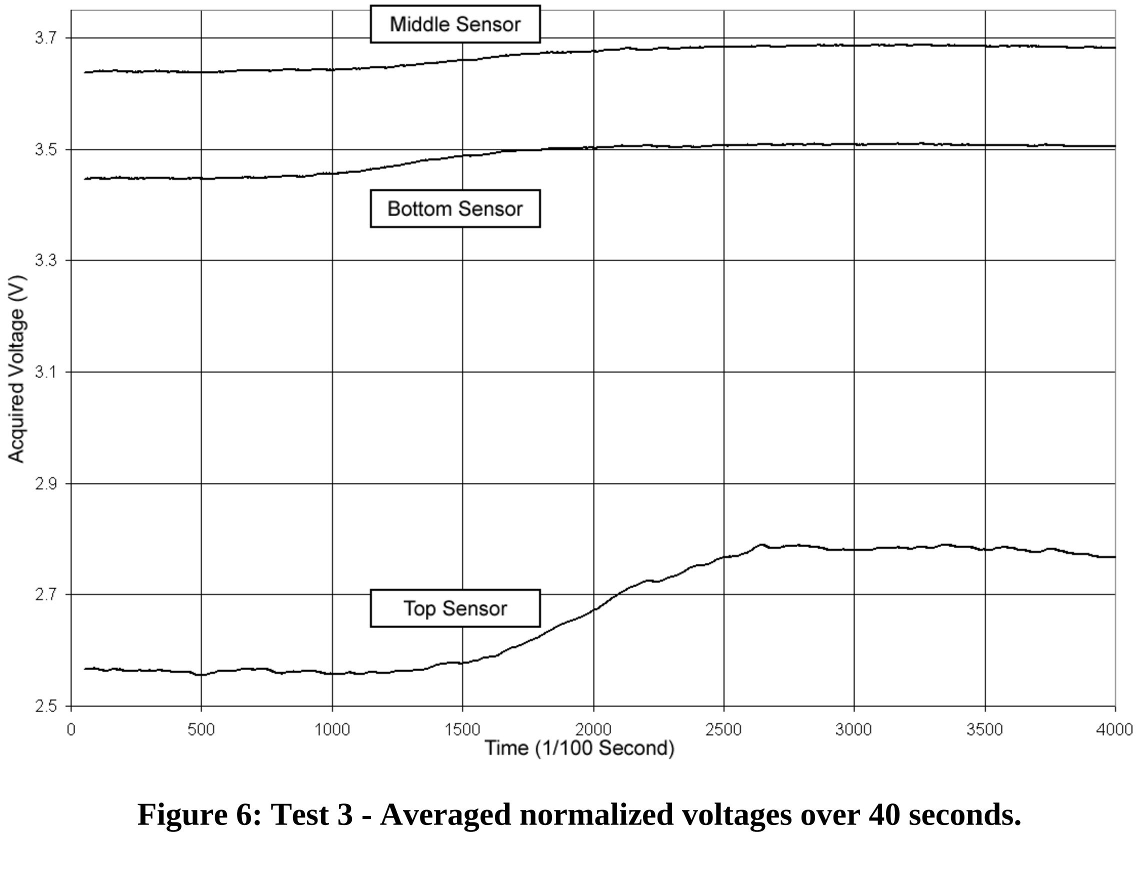


Figure 6: Test 3 - Averaged normalized voltages over 40 seconds.

Currently the voltage output from the sensors has been normalized, but not expanded to the maximum ADC range of the SACS. This explains the difference in voltage range for each sensor. However the information provided by the sensors is still valuable because it can be clearly seen that the molten aluminum has a direct effect on the sensors, and that this data can be acquired by the SACS.

7 - Discussion

7.1 - Suggested Changes for Future Revisions

The SACS *did* successfully record data and export it into an easily usable format. However, in the process of implementation, it became apparent that certain modifications of the original goals had to be considered.

One issue is that this system is dependent on a PC for functionality, and does not provide real-time feedback. Unfortunately, the SACS could not become a stand-alone system because there is no integrated power supply. Additional problems were encountered when trying to interface with the memory module. Because of these hardware related problems, the software was written to run on a PC.

The inability to display and provide “real-time” feedback is primarily due to the fact that there were specifications or guidelines provided for the SACS output. The pump control system which is currently in place is proprietary and does not allow for custom control of the molten aluminum flow rate. Due to the proprietary nature of this system it cannot yet be interfaced with or replaced.

Another reason for the lack of feedback is the inability to store and compare profiles. This problem is primarily dependant on the problems which were encountered with the memory module. As outlined below there are many more aspects of this system that need to be corrected, and several potential future changes for the SACS have been developed.

7.1.1 - Hardware Suggestions

There are several changes necessary that would allow the SACS to become a complete and stand-alone system. These revisions would include the addition of an integrated power supply, fixing problems with input signal clipping, properly interfacing with an appropriate memory module that will be discussed later, proper input protection, EMI/RFI (electromagnetic and radio frequency) shielding, and the ability to give visual and signal feedback.

7.1.1.1 - Integrated Power Supply

Currently the SACS is powered by the 5V source provided by the USB interface with the computer, which in turn creates a dependency of the system on this interface. In order to make the SACS a stand-alone system, it is essential for it to have its own power supply. This source was created early on in the development of the system however, and due to size constraints and safety it was left out of the first revision of the system. This power supply had a 12V output and had a module to create a 5V rail. The power supply used 120V 60Hz AC (alternating current) from a wall outlet, utilized a 10:1 voltage ratio transformer, and a pre-packaged full-wave bridge rectifier together with a large capacitance; they created the 12V DC (direct current) rail. This node connects to a resistor with a 5V Zener diode connected to ground, thus creating a 5V rail. This is a very simple way to create a 5V reference, but with some tuning of the resistor, it is acceptable to use this method up to 100mA. If more current is required, a LDO (linear drop out regulator) may be required, or if even more power is required, a POL (point of load) buck converter may become necessary. These parts are much more expensive than the resistor and zener diode solution, and also require more space and increased number of components.

7.1.1.2 - Unity Gain Buffer Modifications

The major purpose of the buffer was to create a high impedance input to the SACS so it would not load down the sensor output. This buffer was designed to have the same voltage output as it recieves on the input. It was designed using a LM348 quad op amp package as seen in Appendix B. Only after a great deal of testing of this unity gain buffer section was it discovered that this Op-Amp does not work rail to rail. This presents a problem because of the fact that the SACS is supposed to be accurate from 0 to 5V, and the rails are set at 0 and 5V. Our true operating range after the buffer was about 0 to 3.5V.

To solve this problem, there are multiple solutions. The first possible solution is to get a different Op-Amp that has a better operating range. Different types of Op-Amps were tested, including a more expensive CMOS (complimentary metal oxide semiconductor) Op-Amp. The output of this amp was able to reach the high voltage rail at 5V, but was unable to reach below 1.5V. Other Op-Amps had similar problems and all the ones tested could not truly reach rail-to-rail signal levels.

The next possible solution is to increase the high voltage Op-Amp input to a level high enough that clipping would not occur. This would require putting in a boost converter that could change the 5V rail from USB to a higher voltage. A boost converter is also required if there was an integrated power supply. It is true that the power supply creates a 12V DC rail. However, ideally the 5V rails would be combined so that the system could use either the USB power or integrated power. This combining process is most easily done with two diodes where the integrated and USB 5V rails attach to the system rails. These diodes create .7V drop, thus lowering the voltage rail to 4.3V, and the output range of the unity gain amplifier to zero to 2.8V. Again in this case, a boost converter would be required to run the Op-Amps at unity gain. In the first revision, there was no integrated power supply, no diodes, and no boost converter; however there was no problem because the sensor output range was within the 0-3.5V range.

The third possible solution is to give the buffer a gain of less than one; for example a gain of half. In this case with an input of 0 to 5V, the output would be 0 to 2.5V. This solution seems the most viable because it solves the issue of signal clipping with only the use of two resistors. Of course this does change the signal, but it can easily be doubled later on digitally. The problem is that Op-Amp in non inverting configuration cannot have a gain of less than one, and the inverting configuration would essentially flip our signal upside-down. This means that if Op-Amps were used as a negative gain stage, another negative gain stage would need to be present to correct the signal distortion. This would lead to more unnecessary added parts and the usage of even more space.

It seems that the best method is to create a simple resistor divider network which would be placed in front of the Op-Amp buffer to divide in half. This is of course assuming that the current in the divider is significantly larger than the input current into the Op-Amp buffer. If the sum of both resistors of the divider is 5000Ω, at maximum the current through the divider would be .5mA, which is over 1000 times greater than the maximum input current into the Op-Amp. If the Op-Amp did require the maximum

0.5μA, the error would be 2.5mV, which is acceptable due to the low probability of occurrence, and the amplitude of noise on the signal. The modifications to the input modules have been drawn, and added into Appendix B

7.1.1.3 - Reference Voltages

The current A/D (analog to digital converter) reference voltage is internally set to 5V. This means that when polling the inputs, if five volts or more is present, the digital output is 1023(10 bit binary max). If the input were 2.5V the digital output would be 511, and if the input were 0V, the output would be binary 0. If the divider module were implemented, the system input would be halved before reaching the A/D. If the reference voltage were left alone, the digital output range would be only 0-511 which uses only half of the 10 bits, implying that the resolution of the SACS is halved. For this reason it is essential to set the reference voltage to scale the A/D digital output. If software enabled, pins four and five can be tied to ground and a 2.5V reference (respectively), setting the A/D converter so that 2.5V in is digital 1023 out. This solves the scaling problem and can be easily done with the addition of one line of code, and the use of a 2.5V Zener diode and a resistor. Again this modification can be seen in the revised schematic in Appendix B.

7.1.1.4 - EMI/RFI Shielding

In the first revision the only shielding that was implemented was the intrinsic shielding of the coaxial cables. The importance of shielding has already been discussed, but was not implemented in this design revision due to the fact that it is device protection and was not a priority compared to the basic functionality of the system. The shielding process includes creating a complete layer of metal mesh or sheet to surround the internal components of the system. This shield is tied to ground, which is also connected to the shielding of the coaxial cables, and ideally the communications cable of the USB interface. Obtaining a shielded USB cable was also considered inessential for functionality testing, however should be considered for full EMF shielding.

7.1.2 - Memory Integration

Throughout the development of the SACS, there has been a critical need for the system to incorporate some form of memory interface. When it was decided that the

UBW was going to be the heart of the SACS it was also decided that the DOSonCHIP module would be used for data storage.

Through many hours of frustration it was discovered that the DOSonCHIP module could not perform as desired. The problem lies in an approximate DOSonCHIP write limitation of 480Bps. Given the fact that one ASCII character is a byte, and that each sensor uses a minimum of 5 ASCII characters, plus a default 3 characters per line, (so for 3 sensors, 18 bytes) a maximum data-write speed of 26sps reached for only 3 sensors. This limitation only gets worse with each additional sensor.

Although this sampling rate is theoretically well higher than needed for the current application of the SACS, it lends to the idea that another module might be better suited. However, other modules may be subject to the same limitations, and thus it might be required to develop a new module and/or interface to achieve what needs to be done. This module was scrapped from the SACS after large amounts of effort went into debugging the interface with little success in increasing its speed.

7.1.2.1 - Feedback

The SACS was successful in providing a file that could be imported and turned into a graph in Microsoft Excel. Although this does not meet the ideal feedback that was desired at the start of this project, the proprietary nature of the pump system made it impossible to determine what kind of system feedback was necessary for the SACS to produce. This was discovered early on, and it was determined that it was acceptable to only provide a real-time visual feedback.

Feedback was to be in the form of either an LED error indicator, or some form of a real-time software graph which could show what the SACS was receiving as inputs. This is because there is no longer an ideal profile to compare against, nor sufficient device memory to store it in.

There was significant progress made in creating a graphical display through Visual Basic. This graph had the ability to display all of the proper points from each of the three inputs on a proper scale. Unfortunately points from input one were connected to points from input two and points from input three.

After many hours of debugging this graphing module was finally taken out. At this point it was also decided that future revisions should not be have to be reliant on the use of a PC for feedback. It is recommended that in future revisions the LED error display be implemented, or potentially an LCD screen could be included to display a graph of what is currently being observed by the SACS.

Along these lines, it is also recommended that when given proper specifications, an analog feedback signal ought to be created. This step should be easy and would most likely be a scaled analog version of the control signal for the LED display. This would need to have output buffers so that the signal is not loaded down by the power draw of the pump.

7.1.2.2 - Microchip changes

The current revision of the SACS uses a PIC18F2455 which works for the level of complexity that is currently implemented. However, it may be necessary to use a different microchip for later revisions. Currently there are only a few operations that are being performed by the PIC18F2455. The current commands are used to poll the A/D registers, preparing the ASCII representation of the binary analog signal value. This is then sent over USB to the PC which in turn tells the PIC to repeat this process up to 100 times a second.

In future standalone revisions, the processor will be required to do much more complex computing. The processor will need to interface with a memory module, compare the sensed value to an ideal profile, and then calculate some form of a control signal.

This control signal will most likely be in the form of an analog voltage, and would require a Digital to Analog (D/A) converter.   This alone is possibly enough to limit any further work with the PIC18F2455.

Possible main-system alternatives include the Omilex ADuC-MT7020 Development board. However, this would still require the integration of an external memory module.

7.1.3 - Software Suggestions

7.1.3.1 - Change of Language

When programming the GUI of the SACS, VB.NET was chosen as it is ideal for rapid prototyping. This allowed for a GUI that is simple and functional. However by system resources standards it was not efficient. For future revisions a different programming language could be the key to improved performance. A more system based language like C++ could provide the efficiency for the SACS so that it could be used cross-platform. Also switching to a non-Microsoft developed system would allow the SACS to be operated independent of operating system constraints.

7.1.3.2 - Provide Active Visual Feedback

Currently the system provides the ability to do post-process analysis on a fill of a cast only momentarily after each test is complete. Ultimately real time analysis and visualization of the data collected by the system would be ideal to provide engine block manufacturers the best quality control.

This could be done via a visual alert that would identify if the item being cast was being poured correctly or whether it should be scrapped from the line. If strict efficiency was needed a simple "Go" or "No Go" notification could be given to the manufacturers. Whatever the visualization is, real time analysis of the data is absolutely necessary to improve the QA of each cast made.

7.2 - Future Implications

7.2.1 - Extended Test Results

The scientific community has hardly looked at the study of the electromagnetic properties of alloys during phase change. It has been suggested that through the use of electromagnetic sensors the SACS could be used to continuously analyze the EM properties of the molten aluminum as it cools in the mold.

In fact, during our testing at MCT we recorded data for 2 minutes and saw a fluctuation in the electromagnetic potential of the aluminum (as described in the Results section 6.1.2). This could prove that there is some promise in future research.

7.2.2 - Pump Control System

It has proven during the course of this MQP that the SACS can take in data and store it to be looked at just moments after processing. The UBW heart of the SACS is quite capable to providing a voltage output and has even been used to light up two different LEDs dependent upon the voltage level being sensed.

Since there is a direct correlation between height and voltage provided by the sensors, an algorithm could be developed to light a series of LEDs identifying the exact height of the molt in the cast. The next step would be to take this output and link it to a molten aluminum pump so the pump could more accurately be controlled.

7.2.3 - Use of UBW in Other Educational Pursuits

Through the course of this MQP we have found the UBW development board to be a useful tool. It is well worth its price ($30) as an easily programmable PIC controller. It has many advantages over just using a PIC. It is inherently USB compatible, and can be programmed via the USB as well. Within an hour of opening the package we were able to send commands to the UBW so that we could read the voltages that were on the input pins.

It should also be noted since we have been developing with the UBW another MQP group has explored to it as a possible development solution for another data acquisition problem.

7.3 Use of the DAQ and LabView

The advent of using the LabView in conjunction with a commercially bought data acquisition device, the NI-6008, came late in development of this project. As stated earlier in this report, we were able to make very functional programs using the LabView interface including a proof of concept height monitoring and flow rate monitoring program.

These programs were simple to create, identifying an important difference between the customizable device that was produced in the course of this project and an off the shelf device and design suite. The SACS device has a total cost of about $40 and the programming software used is commonly available. The NI-6008, on the other hand, has a price of $150 compounded with a LabView design suite licensing fee of $1199 for a base setup to $4099 for the professional edition which allows the exportation of all programs made in LabView to stand alone executable files.

Clearly the total production costs of the SACS device are a lot lower than anything that may be developed using NI devices and software. Also since the SACS device was developed in-house it is completely customizable. The NI device and

LabView have the advantage of development speed because all of the groundwork of class development and other object orientated design schemes has been completed.

This became clear when we created two basic programs in a matter of hours, as opposed to the months it took to create and develop the SACS device. The decision of creating an in-house solution or outsourcing to solve a problem is always a compromise of cost versus time. The SACS device still needs more development time in order to get to the functionality that was created using LabView. However, even with this additional time it still costs thousands less to develop. The choice of using commercial solution as opposed to in-house solution is solely in the customers' hands, they are the ones with the schedule, and the financial resources.

Appendix A: Personal Financial & Safety Considerations

Financial

The funding of this MQP was done solely through the ECE Dept MQP allowance. The allowance allocates $150 to each student in an MQP team. Combined the MQP team had a budget of $450. The budget was used to purchase all the essential parts for development of the SACS and the travel to and from MCT.

The most expensive part of the SACS system is the UBW. Two were purchased from SparkFun.com at a total cost of $64.50 including shipping and handling. The DOSonChip system was purchased for development of the system but was not utilized due too technical restraints. It was purchased at a cost of $40. The rest of the components that were bought include resistors, diodes, capacitors, enclosure, and BNC connectors. The total purchase price of those items was $30.

The travel expenses that needed to be considered for this MQP came from the trips to MCT in Milford NH. The total miles spent traveling to MCT during the course of this MQP was 275 miles. At a cost of 37 cents per mile the estimate travel cost during the course of this MQP was $101.80.

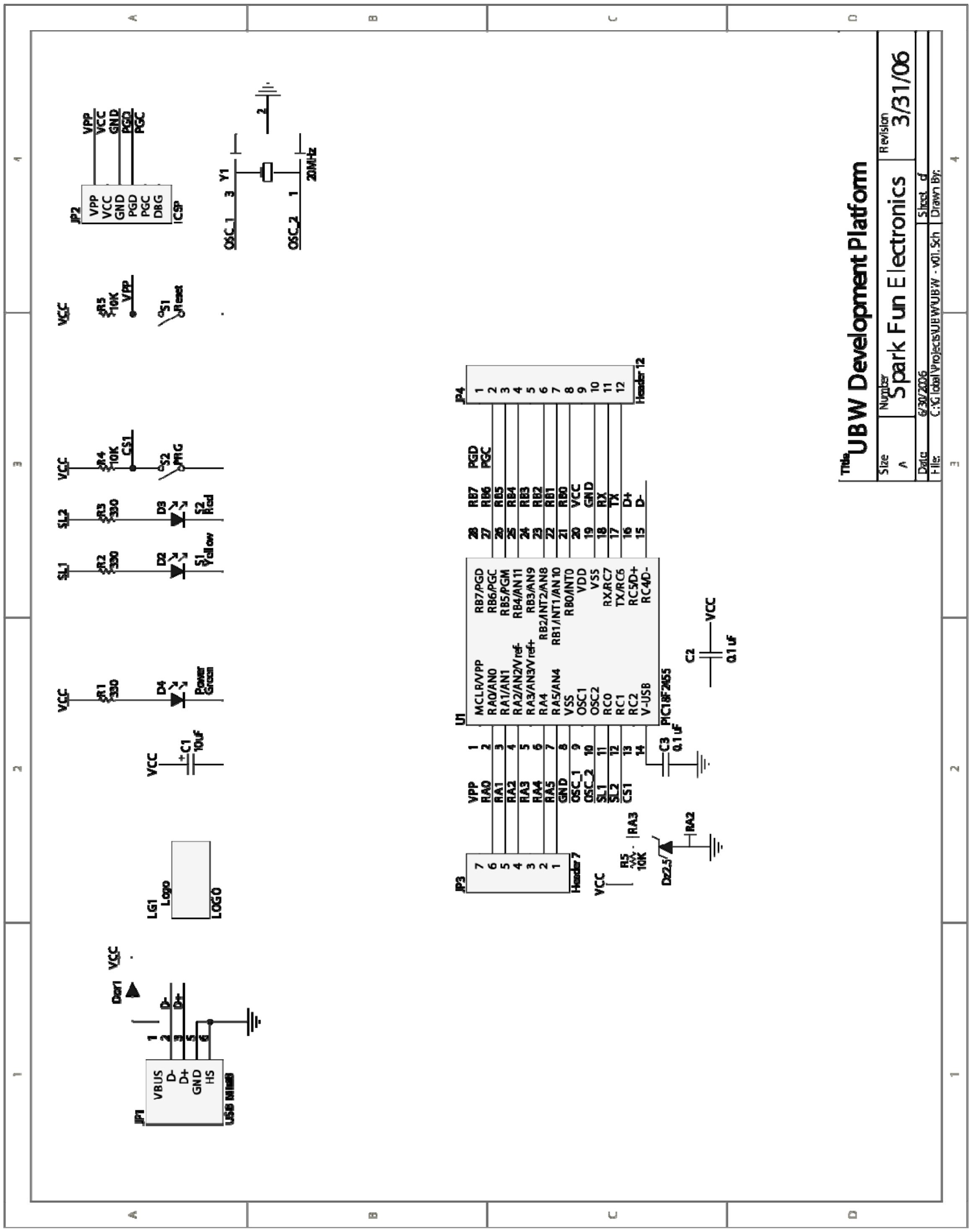
Safety

During the process of testing the SACS in an industrial setting, safety measures were followed in order to comply with OSHA standards and release MCT from a limited spectrum of liability.

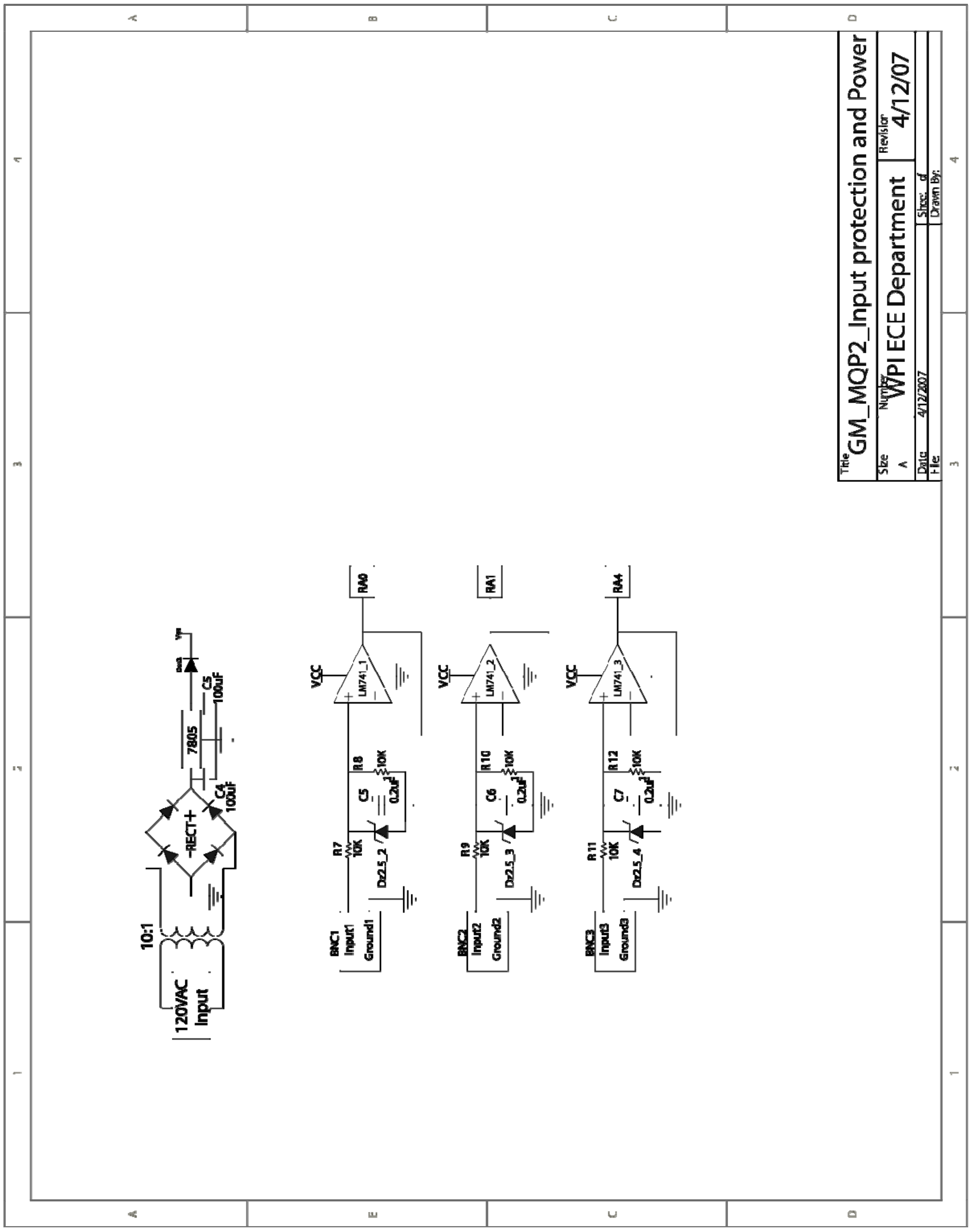
All participants within the SACS testing were required by MCT policy to wear eye protection while on the manufacturing floor. Additionally, a safe distance was determined at which all non-workers would have to be located during the test procedures.

Appendix B: Board Layouts & Schematics

Revised UBW Schematic



SACS Input Protection Schematic



Appendix C: Source Code

All source code is attached to this document in the accompanying CD.

Appendix D: Further Explanation of UBW Class Methods

Method List

Properties:	Class Specific Methods:	General Utility Methods:	Command Methods:
• Port Question • Connection Verification • RS-232 Carry-Over Properties	• Find COM Port • Connect • Serial Open • Disconnect	• Toggle LED • Toggle Pin • ID Response	• Configure • Write All • Read All • Write Pin • Read Pin • Pin Direction • Version • Reset • Memory Write • Memory Read • Analog Sample • Timed Sample

UBW Class Properties

Port Question

The UBW class contains a property called PortQuestion() that can be used to get or set the unique identifier string. By allowing this, it is possible to have numerous different SACS devices attached to a single computer, each potentially recording values from different input devices.

The default identification string is set to match the default firmware version. This string currently is “UBW.D.130”. The computer will only make a connection to the UBW if the port-question matches the unique id (currently the version string) of the corresponding UBW device.

Connection Verification

The UBW class contains a property called conxd() that returns a Boolean value used to identify the status of the current UBW connection. If the device currently has an active RS-232 connection, this property will display true. If inactive, this property will display false.

This property is read-only because the function of connecting/disconnecting is left up to a different function.

RS-232 Carry-over Properties

In order to open an RS-232 connection, there are six necessary parameters: baud-rate, bit-stream length, parity bit, number of stop bits, buffer length, and (most importantly) which com port.

To increase flexibility of the UBW class, these properties of the RS-232 connection are abstracted by getter & setter functions at the top level. Each of the properties is used in activation of an RS-232 connection. This means all properties must be set prior to opening a new connection.

UBW Class Functions

Private Function: Find COM Port

This function, findComPort(), is designed to systematically search through 100 COM ports/Virtual (VCOM) ports. It will return an integer port number upon successful verification of the UBW's unique identification.

The location of the UBW is determined by looping through an excessive number of COM & VCOM ports. On each port, a connection attempt is made to access the UBW. Failure to connect determines the absence of a UBW. Connection however, does not determine whether or not to verify a UBW existence on a particular port.

Because there is the possibility to connect multiple UBW devices to the same computer system, the port question (See "*Software Development - UBW Class Properties: Port Question*") is used to verify that the UBW currently attached to the computer system is the correct one for the desired system performance.

When the UBW firmware version is verified against the port question, the port number is recorded, and returned from the function.

If there are multiple UBW devices connected with identical firmware names, they will register on two different VCOM ports. For the simplification of the project, only the first of identical UBW devices will be connected to. However, once a connection is made to a port, another connection attempt to the same port will result in a failure. This would cause the findComPort() to return a '-1' while looking to make a new connection to a UBW with identical specifications and versioning.

Public Function: Connect

This function is actually the abstract interface to the findComPort() function, allowing a connection to be made in a single function call, given setup of or defaulted connection properties.

Public Function: Serial Open

This function and its overloaded version allow for a serial port connection to be made. For the most part, this connection would be used for internal UBW class functionality. However it does allow for access to the serial connection directly from the top level.

The first version of this function operates simply by using the default UBW class values for COM port, baud rate, bit stream length, parity bits, number of stop bits, and buffer size. The overloaded version takes in an integer value for connecting to a specific

COM port. Both functions return Boolean true upon successful connection and Boolean false otherwise.

Public Function: Disconnect

This function is a direct abstraction of the RS-232 connection's close() function. This function has the added feature of changing the UBW class's "connected" Boolean to false.

This function returns true if the disconnect is possible and false otherwise.

UBW Class General Utility Methods

Toggle LED

This is used to toggle any of the UBW's on-board LEDs. These LEDs are by default connected to the PIC Microprocessor's Port C. The direct functionality of this method is provided by calls to read_pin() and write_pin().

By reading the value of Port C (where the LEDs are connected on the default UBW board), the toggle function merely switches the specified LED to its opposite state. Thus if the LED is on, it is turned off, and vice versa.

Toggle Pin

This method functions similar to the Toggle LED method. The difference in operation is only in that a specific port must be passed as a parameter. This port is then used as the address of the pin to be toggled.

As with the Toggle LED method, the Toggle Pin method functionality comes from reading the desired port and pin, then writing the opposite value to that pin.

ID Response

This method is used to identify what the UBW did in response to any particular command. This method works only because every command method of the UBW class passes the actual response from the UBW firmware through this method.

There are eight command specific success responses, five specific error responses, and a catch-all general error response. The purpose of this method is to translate the otherwise cryptic response of the UBW hardware into a more meaningful response for the interface user.

UBW Class Command Methods

Configure

This is used to configure port a, b and c pin directions and also to decide how many analog pins to initialize. Analog pin initialization is explained more accurately in the appendix as part of the Firmware documentation of the configure command.

The basic functionality is simply to pass the configure command to the UBW hardware and await a response. This requires that an 8-bit binary mask be created for each port on the UBW, a 0-set bit indicating an output and a 1-set bit indicating an input. This number then is converted to a decimal number and passed as the port a, b, and c pin directions.

The analog pin initialization is complex to understand, but basically consists of an 8-bit binary number where a pin is specified as analog by being set to a 1. The stipulation to this is that all pins under the highest pin selected as analog must also be analog. Not adhering to this rule will simply cause the analog equivalent of the digital value of the pin's current output voltage to be presented as the only value on that pin.

This could be used as a desired effect, but is otherwise rather useless, and is simply a limitation to the current firmware structure of the UBW.

Write All

This method utilizes the UBW firmware to functionally select each individual pin on each port, and set them to some specified value. These values are provided as 3 decimal-equivalents of 8-bit binary numbers, one for each port.

Read All

This method is an enhanced form of the Read Pin method, and provides a basis for checking the value of multiple pins at once. All ports and pins are read, and returned as a string containing the value of each pin on each port.

Write Pin

This method changes the state of a single pin on a specified port. This is a digital method, and does not provide any useful representation for analog port data.

When run properly, this method will change the value of the specified pin on the specified port to the specified value (either 1 for on or 0 for off), and will return a command response to identify successful command execution.

Read Pin

This method reads the value of a single pin on a specified port. This is a digital method, and does not provide any useful representation of analog port data.

Simply by providing a string port name and integer pin number, this method will respond with a string containing either a 1 (for on) or a 0 (for off).

Pin Direction

The Pin Direction command changes an input pin to an output pin or vice-versa. This is a simplification to a single part of the configure command, and allows for a single pin to change between I/O states.

This method requires a string port name, integer pin number, and integer direction. The port name specifies the port on which to change the specified pin. The direction is provided as either a 1 (for input) or a 0 (for output).

Version

The version command is used to send back the Firmware coded Version string. This string is beneficial in identifying application specific UBW devices, and is used in the port question method for identification of the UBW's COM port.

This method requires no input parameters, and returns a string containing the firmware coded version.

Reset

This method resets all pins and ports to default values. This eliminates any previous configuration that was operating on the UBW.

This method requires no input and provides no output beyond the command response to identify successful command execution.

Memory Write

This method writes a value to memory at the address provided. The address provided must be a valid PIC memory address, and all values to be written to memory must be in integer form.

Memory Read

Reads the value stored in memory at the address provided. The address must be a valid PIC memory address. The value is returned as a string, and must be cast to the appropriate data type on the interface side.

Analog Sample

The Analog Sample command samples all inputs specified as 'Analog' by the Configure command. These values are returned as a string in which each analog port is represented by a comma-delimited, decimal equivalent, 10-bit number.

Timed Sample

This command allows for a single call to the UBW device to return streaming samples from the analog ports. This runs equivalent to the Analog Sample command on the UBW, but returns based upon a millisecond resolution and an I/O direction. The resolution is currently limited to one sample every ten milliseconds, or equivalently 100 samples per second.

On the UBW firmware side, the benefit of this method of analog input reading is that it eliminates successive calls to the UBW, effectively reducing USB bandwidth usage by half.

As with the Analog Sample command, these values are returned as a string in which each analog port is represented by a comma-delimited, decimal equivalent, 10-bit number.

Appendix E: SACS User Manual

Sensor Acquisition and Control System (SACS)

User Manual

Introduction:

Hello and Welcome to the Sensor Acquisition and Control System (SACS) User Guide. This Guide was created to give you all the information you need in order to use the SACS device correctly. Hopefully this device will give you the ability to store any data produced by the Electromagnetic Coil Sensors for future reference and analysis using MS Excel Suite.

Installing:

Before you connect anything to the SACS device you need to install the SACS driver so that the PC can recognize and talk to the SACS device.

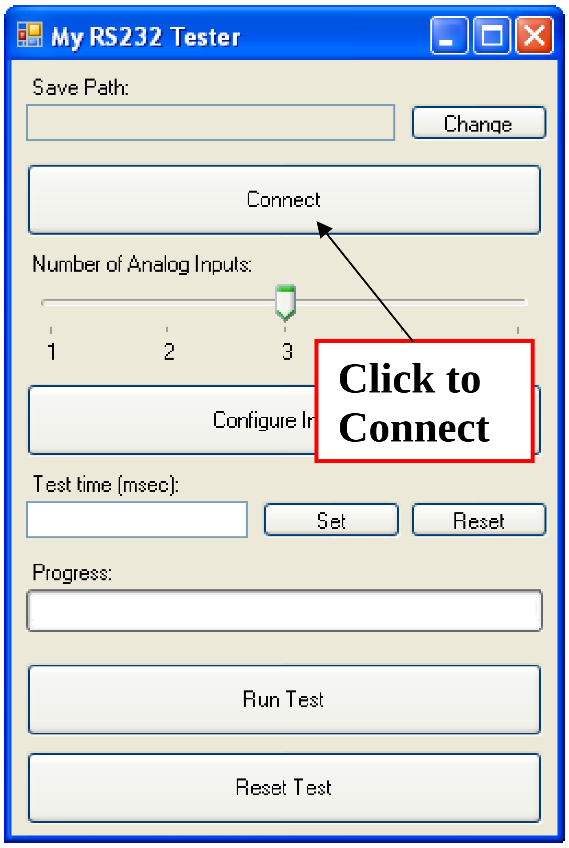
Connecting:

If you have not already please connect the BNC connector ends to both the Sensor Driver Circuit and the SACs Device. This will allow the Sensor Driver Circuit to send information to the SACS device. BE CAREFUL while connecting any hardware and follow the ESD guidelines provided by your Employer.

Once the SACS device and the Sensor Driver Unit have been connected it is time to connect the SACs device to the PC you loaded the SACS program on. Take the USB cable that is connected to the SACS device and connect it to the PC. If the installation of the SACS driver was successful the computer will detect the SACS device as s USB device. Now you can double click on the SACS program icon to bring up the user interface. You can self-test the connection of the SACS device and the computer by clicking on the "Connect" button.

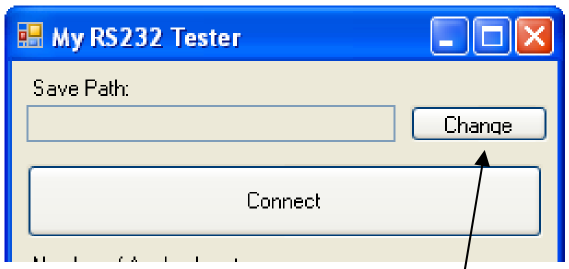
If a successful connection has been made the Connect button will grey out, and be unavailable to be clicked.

If an unsuccessful connection occurred then a dialog box will open explaining so. If this happen try to manually disconnect and connect the USB cable and try again. If a connection still cannot be made, then re-install the SACS driver software.



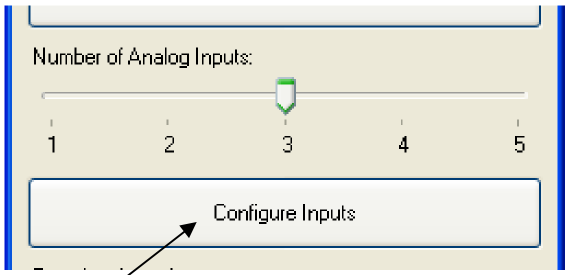
Configuration:

Now that SACS graphical user interface (GUI) is displayed on the windows desktop and the SACS device has been successfully connected to the PC you can begin to setup your first test. The first thing you should do, and make a habit of every time you use the SACS, is to select the directory the SACS program will save all the data collected during the sampling. Simply click on the "change" button next to the directory box and click where you want to save.



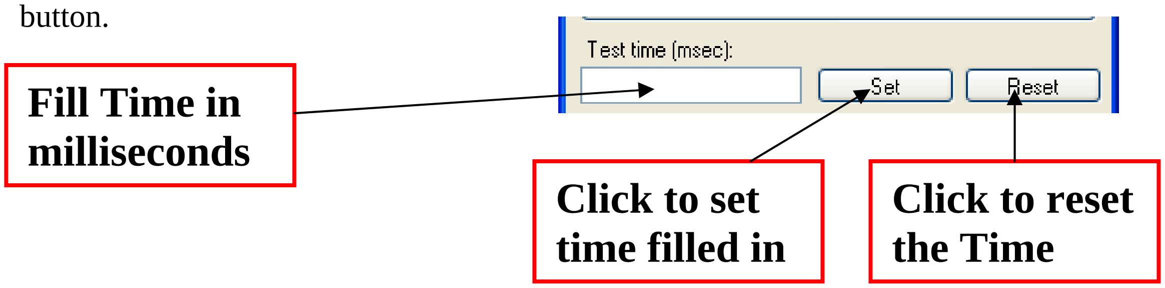
Click to Change Save Directory

Once the location of the data collection has been confirmed you can now select how many sensors you want to take data from. Currently the SACS device has a maximum of 3 sensor connections, but a maximum of 5 can be supported. Simply drag the slider to how many sensors you want. One you are satisfied click on the "Configure Inputs" button.



Click to Set Amount of Sensors

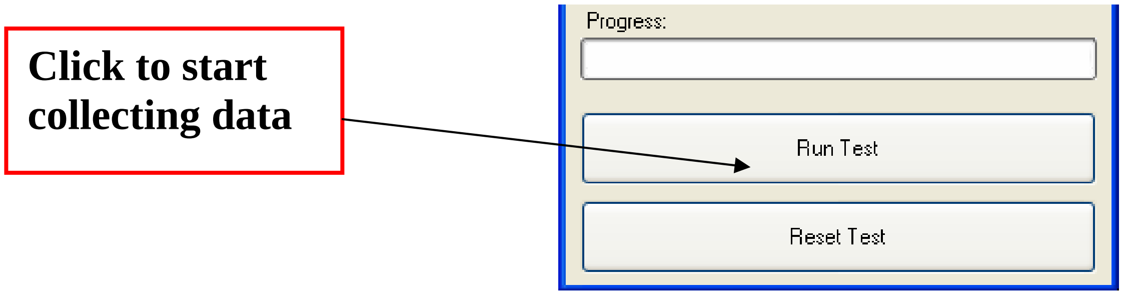
Now you can set the amount of time you want to collect data for. You can do this by entering how much time in milliseconds into the Test time fill in box. Then click the "Set" button. If you decide to change the time before running the test click the "Reset" button.



The test time is the last piece information that the SACS program needs to know in order to run a test.

Testing:

In order to start collecting data from the Electromagnetic Coil Sensors you must click on the "Run Test" button. This will collect data for the specified time from the specific amount of sensor set by you.



Once clicked the "Run Test" button will be grayed out until the test has been completed. You can the find the Excel readable .CSV file in the file you selected with this name scheme:

2006128115316.csv

Where the first 4 digits is the year, then the month, day, and military time.

Post Data Analysis:

Once you have the data file you can open it up with MS Excel. Excel provides you with numerous ways to analyze the data you have collected. To be specific all data files will have all of the sensors data in columns. So the first column of data will be Sensor 1, and etc. You can create a visualization of the data and actually see how the molten aluminum flowed up the cast. Here is an example of what you can do.

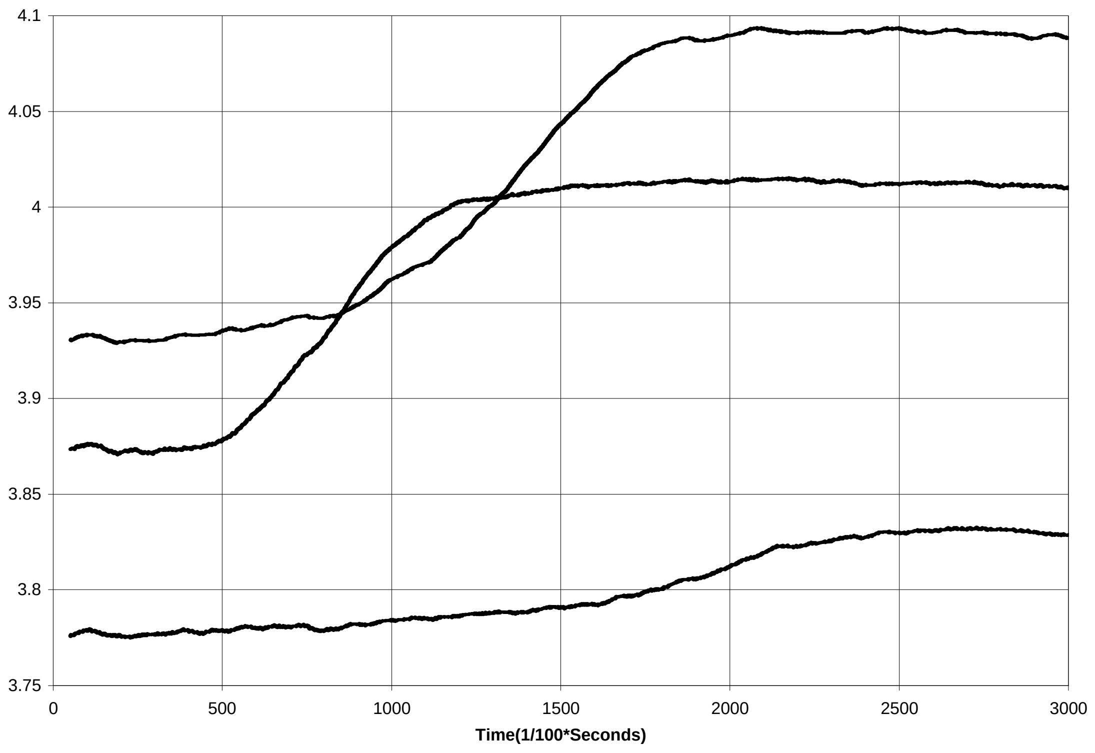


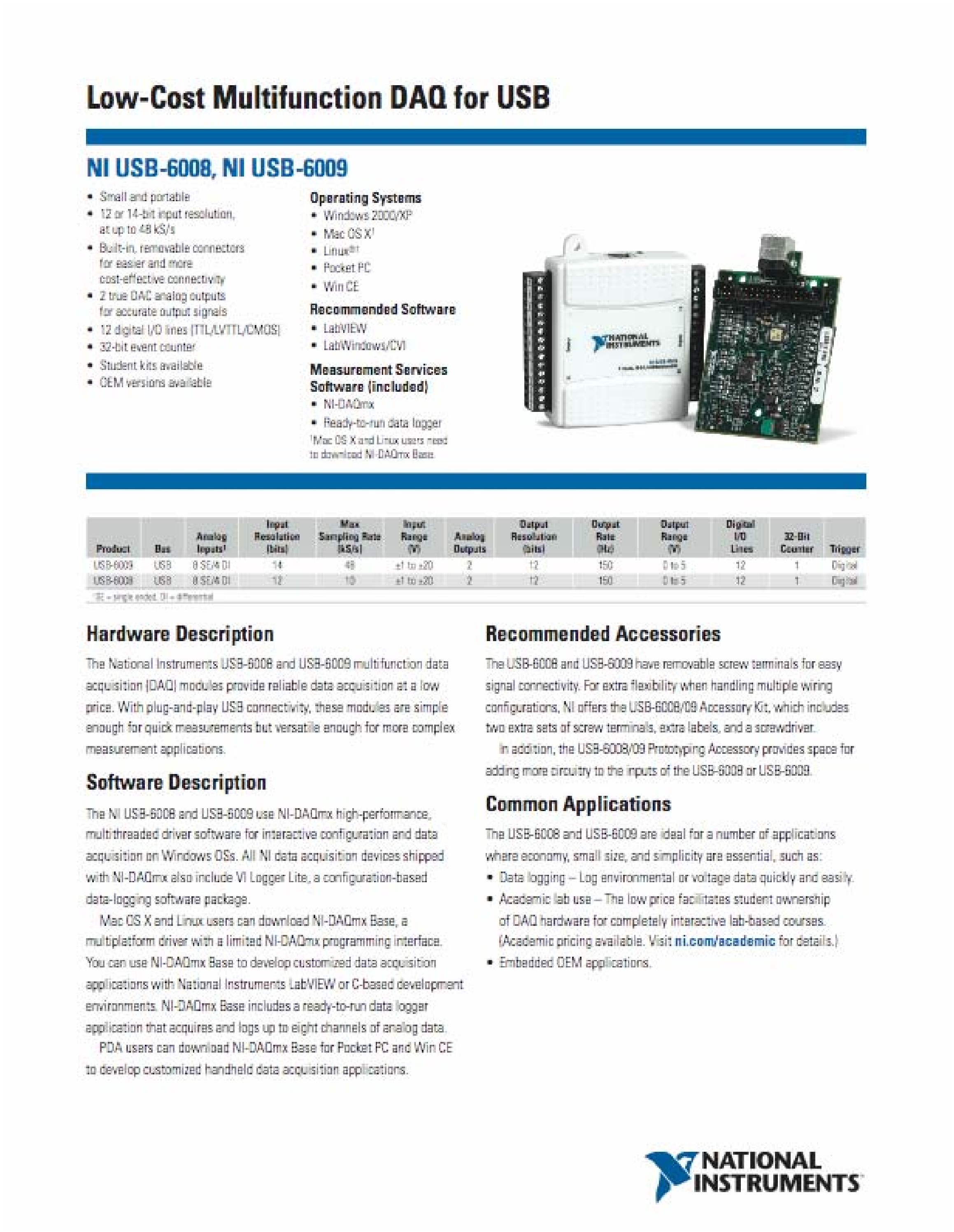
Figure E-1: Post-processed data from 3 sensors

Above is an averaged plot representation of the data that was collected from three sensors over 35 seconds, or 3500ms.

This concludes the use of the SACS device. If you need more specific Information please read the full report entitled Sensor Acquisition and Control System.

Appendix F: Data Sheets for NI-6008 and Test Bench Laptop

NI-6008 DAQ



Low-Cost Multifunction DAQ for USB

NI USB-6008, NI USB-6009

- Small and portable
- 12 or 14-bit input resolution, at up to 48 kS/s
- Built-in, removable connectors for easier and more cost-effective connectivity
- 2 true DAC analog outputs for accurate output signals
- 12 digital I/O lines (TTL/LVTTL/CMOS)
- 32-bit event counter
- Student kits available
- OEM versions available

Operating Systems
- Windows 2000/XP
- Mac OS X[1]
- Linux®[1]
- Pocket PC
- Win CE

Recommended Software
- LabVIEW
- LabWindows/CVI

Measurement Services Software (included)
- NI-DAQmx
- Ready-to-run data logger

[1]Mac OS X and Linux users need to download NI-DAQmx Base.

Product	Bus	Analog Inputs[1]	Input Resolution (bits)	Max Sampling Rate (kS/s)	Input Range (V)	Analog Outputs	Output Resolution (bits)	Output Rate (Hz)	Output Range (V)	Digital I/O Lines	32-Bit Counter	Trigger
USB-6009	USB	8 SE/4 DI	14	48	±1 to ±20	2	12	150	0 to 5	12	1	Digital
USB-6008	USB	8 SE/4 DI	12	10	±1 to ±20	2	12	150	0 to 5	12	1	Digital

[1]SE = single ended, DI = differential

Hardware Description

The National Instruments USB-6008 and USB-6009 multifunction data acquisition (DAQ) modules provide reliable data acquisition at a low price. With plug-and-play USB connectivity, these modules are simple enough for quick measurements but versatile enough for more complex measurement applications.

Software Description

The NI USB-6008 and USB-6009 use NI-DAQmx high-performance, multithreaded driver software for interactive configuration and data acquisition on Windows OSs. All NI data acquisition devices shipped with NI-DAQmx also include VI Logger Lite, a configuration-based data-logging software package.

Mac OS X and Linux users can download NI-DAQmx Base, a multiplatform driver with a limited NI-DAQmx programming interface. You can use NI-DAQmx Base to develop customized data acquisition applications with National Instruments LabVIEW or C-based development environments. NI-DAQmx Base includes a ready-to-run data logger application that acquires and logs up to eight channels of analog data.

PDA users can download NI-DAQmx Base for Pocket PC and Win CE to develop customized handheld data acquisition applications.

Recommended Accessories

The USB-6008 and USB-6009 have removable screw terminals for easy signal connectivity. For extra flexibility when handling multiple wiring configurations, NI offers the USB-6008/09 Accessory Kit, which includes two extra sets of screw terminals, extra labels, and a screwdriver.

In addition, the USB-6008/09 Prototyping Accessory provides space for adding more circuitry to the inputs of the USB-6008 or USB-6009.

Common Applications

The USB-6008 and USB-6009 are ideal for a number of applications where economy, small size, and simplicity are essential, such as:
- Data logging – Log environmental or voltage data quickly and easily.
- Academic lab use – The low price facilitates student ownership of DAQ hardware for completely interactive lab-based courses. (Academic pricing available. Visit **ni.com/academic** for details.)
- Embedded OEM applications.

NATIONAL INSTRUMENTS™

Specifications

Typical at 25 °C unless otherwise noted.

Analog Input

Absolute accuracy, single-ended

Range	Typical at 25 °C (mV)	Maximum (0 to 55 °C) (mV)
±10	14.7	138

Absolute accuracy at full scale, differential[1]

Range	Typical at 25 °C (mV)	Maximum (0 to 55 °C) (mV)
±20	14.7	138
±10	7.73	84.8
±5	4.28	58.4
±4	3.59	53.1
±2.5	2.56	45.1
±2	2.21	42.5
±1.25	1.70	38.9
±1	1.53	37.5

Number of channels	8 single-ended/4 differential
Type of ADC	Successive approximation

ADC resolution (bits)

Module	Differential	Single-Ended
USB-6008	12	11
USB-6009	14	13

Maximum sampling rate (system dependent)

Module	Maximum Sampling Rate (kS/s)
USB-6008	10
USB-6009	48

Input range, single-ended	±10 V
Input range, differential	±20, ±10, ±5, ±4, ±2.5, ±2, ±1.25, ±1 V
Maximum working voltage	±10 V
Overvoltage protection	±35 V
FIFO buffer size	512 B
Timing resolution	41.67 ns (24 MHz timebase)
Timing accuracy	100 ppm of actual sample rate
Input impedance	144 k
Trigger source	Software or external digital trigger
System noise	0.3 LSB$_{rms}$ (±10 V range)

Analog Output

Absolute accuracy (no load)	7 mV typical, 36.4 mV maximum at full scale
Number of channels	2
Type of DAC	Successive approximation
DAC resolution	12 bits
Maximum update rate	150 Hz, software-timed
Output range	0 to +5 V
Output impedance	50 Ω
Output current drive	5 mA
Power-on state	0 V
Slew rate	1 V/µs
Short-circuit current	50 mA

Digital I/O

Number of channels	12 total 8 (P0.<0..7>) 4 (P1.<0..3>)
Direction control	Each channel individually programmable as input or output
Output driver type	
USB-6008	Open-drain
USB-6009	Each channel individually programmable as push-pull or open-drain
Compatibility	CMOS, TTL, LVTTL
Internal pull-up resistor	4.7 kΩ to +5 V
Power-on state	Input (high impedance)
Absolute maximum voltage range	-0.5 to +5.8 V

Digital logic levels

Level	Min	Max	Units
Input low voltage	-0.3	0.8	V
Input high voltage	2.0	5.8	V
Input leakage current	–	50	µA
Output low voltage (I = 8.5 mA)	–	0.8	V
Output high voltage (push-pull, I = -8.5 mA)	2.0	3.5	V
Output high voltage (open-drain, I = -0.6 mA, nominal)	2.0	5.0	V
Output high voltage (open-drain, I = -8.5 mA, with external pull-up resistor)	2.0	–	V

Counter

Number of counters	1
Resolution	32 bits
Counter measurements	Edge counting (falling edge)
Pull-up resistor	4.7 kΩ to 5 V
Maximum input frequency	5 MHz
Minimum high pulse width	100 ns
Minimum low pulse width	100 ns
Input high voltage	2.0 V
Input low voltage	0.8 V

Power available at I/O connector

+5 V output (200 mA maximum)	+5 V typical +4.85 V minimum
+2.5 V output (1 mA maximum)	+2.5 V typical
+2.5 V output accuracy	0.25% max
Voltage reference temperature drift	50 ppm/°C max

[1]Input voltages may not exceed the working voltage range.

Physical Characteristics

If you need to clean the module, wipe it with a dry towel.

Dimensions (without connectors)	6.35 by 8.51 by 2.31 cm (2.50 by 3.35 by 0.91 in.)
Dimensions (with connectors)	8.18 by 8.51 by 2.31 cm (3.22 by 3.35 by 0.91 in.)
Weight (without connectors)	59 g (2.1 oz)
Weight (with connectors)	84 g (3 oz)
I/O connectors	USB series B receptacle (2) 16-position (screw-terminal) plug headers
Screw-terminal wiring	16 to 28 AWG
Screw-terminal torque	0.22 to 0.25 N•m (2.0 to 2.2 lb•in.)

Power Requirement

USB (4.10 to 5.25 VDC)	80 mA typical 500 mA maximum
USB suspend	300 µA typical 500 µA maximum

Environmental

The USB-6008 and USB-6009 are intended for indoor use only.

Operating environment	
Ambient temperature range	0 to 55 °C (tested in accordance with IEC-60068-2-1 and IEC-60068-2-2)
Relative humidity range	10 to 90%, noncondensing (tested in accordance with IEC-60068-2-56)
Storage environment	
Ambient temperature range	-40 to 85 °C (tested in accordance with IEC-60068-2-1 and IEC-60068-2-2)
Relative humidity range	5 to 90%, noncondensing (tested in accordance with IEC-60068-2-56)
Maximum altitude	2,000 m (at 25 °C ambient temperature)
Pollution degree	2

Safety and Compliance

Safety

This product is designed to meet the requirements of the following standards of safety for electrical equipment for measurement, control, and laboratory use:

- IEC 61010-1, EN 61010-1
- UL 61010-1, CAN/CSA-C22.2 No. 61010-1

Note: For UL and other safety certifications, refer to the product label or visit **ni.com/certification**, search by model number or product line, and click the appropriate link in the Certification column.

Electromagnetic Compatibility

This product is designed to meet the requirements of the following standards of EMC for electrical equipment for measurement, control, and laboratory use:

- EN 61326 EMC requirements; Minimum Immunity
- EN 55011 Emissions; Group 1, Class A
- CE, C-Tick, ICES, and FCC Part 15 Emissions; Class A

Note: For EMC compliance, operate this device according to product documentation.

CE Compliance

This product meets the essential requirements of applicable European Directives, as amended for CE marking, as follows:

- 73/23/EEC; Low-Voltage Directive (safety)
- 89/336/EEC; Electromagnetic Compatibility Directive (EMC)

Note: Refer to the Declaration of Conformity (DoC) for this product for any additional regulatory compliance information. To obtain the DoC for this product, visit **ni.com/certification**, search by model number or product line, and click the appropriate link in the Certification column.

Waste Electrical and Electronic Equipment (WEEE)

EU Customers: At the end of their life cycle, all products must be sent to a WEEE recycling center. For more information about WEEE recycling centers and National Instruments WEEE initiatives, visit **ni.com/environment/weee.htm**.

Test Bench Laptop

HP dv2000t laptop specs:
Intel(R) Core(TM) 2 Duo T5200(1.60GHz/2MB L2Cache)
128MB NVIDIA(R) GeForce(R) Go 7200
2GB DDR2 RAM
80GB 5400RPM SATA Hard Drive

Appendix G: References

[1] SUNY Schenectady County Community College –
http://www.sunysccc.edu/academic/mst/ptable/al.html – *Citation*: Lide, David R., ed. CRC Handbook of Chemistry and Physics, 78th Ed., 1997-1998.

[2] Aluminum Association, Inc : (a) Industry Overview (b) Aluminum Products (c) Issues Facing the Industry (d) Transportation Market (e) Castings :
http://www.aluminum.org/template.cfm?Section=The_Industry

[3] Aluminum Association, Inc - Automotive Aluminum : http://www.autoaluminum.org/

[4] Sand Casting - http://en.wikipedia.org/wiki/Sand_cast

[4] Sand Casting Explained – Metal Technologies, Inc.
http://www.metal-technologies.com/SandCasting.aspx

[6] Precision Sand Casting by James Van Wert, Amcast Industrial Corporation, Dayton, Ohio 45959

[7] USPTO Patent Number 4,693,292 – John Campbell (Worcester, GB) – Sept. 15, 1987 – *Online*: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=40&f=G&l=50&co1=AND&d=PTXT&s1=Cosworth&OS=Cosworth&RS=Cosworth

[8]Office of the Deputy PM: British Geological Survey
http://www.mineralsuk.com/britmin/mpfsilica_sand.pdf

[9] UBW (USB Bit Whacker) – Brian Schmalz
http://greta.dhs.org/UBW/

[10] SparkFun Electronics
http://www.sparkfun.com/

Figure 1:
http://en.wikipedia.org/wiki/Image:CoreBoxPatternCoreCasting.jpg

Figure 2
http://www.makinamuhendisi.com/images/Castin56.gif

Appendix H: Acknowledgments

We would like to acknowledge the individuals who have contributed to the development of this project.

- Professor Advisors Reinhold Ludwig and Sergey Makarov for their integral involvement throughout, and for providing us with a very unique opportunity to work hands-on with industry professionals.

- Scott Biederman for allowing us to participate in on-site testing of the sensors and the SACS

- Brian Schmalz for the original concept and design of the USB Bit-Whacker, and also for detailed communications over the viability of using the UBW with the DOSonCHIP module.

- Brian Foley for his continued work and dedication to the electromagnetic sensors, and for providing us with the ability to run a very iterative and incremental design process through direct testing with the sensors themselves.

- Hans Jenson and Sean Hallinan for the development of the resonance circuit, without which we would never have been able to record such accurate and attractive data.

Special thanks to all others who supported the development of the SACS throughout the term of this project.