Embedded Measurement and Control Applications Utilizing Python on the Pocket BeagleBone

Mr. Stephen A. Strom, Penn State Erie, The Behrend College

Stephen Strom is a lecturer in the Electrical and Computer Engineering Technology department of Penn State Behrend, and holds a B.S. in electrical engineering from Carnegie Mellon University. His career includes over thirty years experience in designing and programming embedded systems and has multiple patents for both hardware designs and software algorithms

Marius Strom, Saint Francis University

Br. Marius is a Franciscan friar of the Third Order Regular of Saint Francis of Assisi, and has worked as an engineering laboratory instructor at Saint Francis University since 2017 after earning an MS in Aerospace Engineering from the University of Maryland.

Embedded Measurement and Control Applications Utilizing Python on the Pocket BeagleBone

Abstract

Open source Linux platforms can be used in a variety of academic courses. By choosing Linux, the instructor can utilize inexpensive hardware to demonstrate programming concepts, hardware interface, algorithms and data analysis. Many low cost embedded Linux boards (such as BeagleBone and Raspberry Pi) contain a variety of busses (SPI, I2C, CAN), general purpose I/O pins, serial ports, PWM outputs, and analog inputs, making them an appropriate choice for a course that has a data acquisition focus. Potential courses include measurements and instrumentation, wireless communications and control systems.

This work demonstrates usage of the Pocketbeagle, an inexpensive, Linux-based microprocessing platform, in the context of a data acquisition and analysis course found in a General Engineering program. This course contains elements of Python-based software development but emphasizes software design and development with respect to an I/O interface for data applications. In addition to this course, it will be demonstrated that the Pocketbeagle is capable of filling various needs in other lower- and upper-division courses.

Labs covered with the Pocketbeagle include digital and analog I/O operations, PWM and UART interface lab projects, all using Python programming. Several of these labs will be discussed in this paper, along with schematics, configurations, and results as well as an assessment of how well the students were able to achieve the course goals.

Introduction

In a General or Mechanical Engineering degree, there are many classes that incorporate microprocessors / microcontrollers as part of their curriculum. The format for each of these classes are similar (in curriculum) in that their end goal is to teach Python programming, as well as embedded hardware and applications.

While educational philosophy and pedagogy will vary from one instructor to another, it could be said that most computer courses employ a large amount of hands-on lab material and selecting a proper embedded processor/operating system can greatly improve the outcome and success of the course. In general, the preference is to use a platform that has:

- **Wide industry acceptance (usage after graduation):** This allows the students to leverage their knowledge into better/more advanced job positions.
- **Development tools that are quickly installed and are easy to use:** There are always questions about compiling, downloading and debugging, and it is important to have local (and online) help tools that can provide solutions to common problems.
- **Low cost development tools, boards, or kits:** A strong suggestion is to have the students purchase their own components. Students that own their own board take better care of them and are more likely to use them in other projects.

- **Compatibility with programming languages of interest:** while programming languages are relatively easy to pick up after one is learned, it is desirable to give students multiple opportunities to hone their skills on a language that is widely adopted by industry.

In addition to these requirements, a microcontroller also needs to fulfill the outcomes required by the course. As this study follows the implementation of a microcontroller in the context of a 2-credit junior-level introductory instrumentation laboratory (ENGR335), examples from this course will be used for discussion. For reference, this class is designed to familiarize the students with basic engineering instrumentation while giving them hands-on experience with basic data collection and reduction using a microprocessing platform. As such, the data acquisition tasks in this class dictate a certain set of hardware requirements said microprocessing platform, such as a need for analog and digital I/O, serial communication ports/busses, and sufficient on-board memory for data collection and manipulation.

To provide for these requirements as well as the aforementioned considerations, the Linux-based Pocketbeagle was selected for use in this course. Produced by Beagleboard as of 2017, the Pocketbeagle presents an extremely small form factor computer with processing and I/O capabilities comparable to larger, more common boards (e.g., Raspberry Pi, Beaglebone Black, etc.), and at a lower cost. This paper will describe the successful usage of this board in the context of ENGR335 and other embedded systems courses, as might be found in a General or Mechanical Engineering program.

Course Overview

The General Engineering curriculum at Saint Francis University is set up so that programming and embedded circuit design is taught in the sophomore and junior years of the program. To accommodate scheduling needs, the basics of Python programming are taught in a Sophomore Fall programming laboratory (although this was not a pre-requisite for ENGR335 in the discussed offering of the course), and instrumentation is discussed in a Junior-level Spring laboratory course. The purpose of the instrmentation course is to introduce the students to common forms of engineering instrumentation, including laboratory experiences utilizing a programmable microcontroller to gather and analyze data. This experience culminates in a real or designed term project, based on instrumentation needs in other courses or research projects.

The course meets for two 50 minute periods per week to allow for traditional lectures as well as in-class design and experimentation. Students have access to both sensors and microcontrollers outside of class meetings as well to facilitate their use in homework assignments and projects. Student understanding is assessed through homework (e.g., small, short-term data acquisition tasks), larger, multi-week lab projects, and a term project. As both the Pocketbeagle and Python have a plethora of freely available online resources, the required course text focuses primarily on data acquisition methods and theory.

Linux Board Selection

There are many embedded Linux development boards that can be used in a university setting. An abbreviated list of the boards considered for use the instrumentation class mentioned is shown below:

- Raspberry Pi [2] ($35)
- BeagleBone Black [4] ($55)
- Pocketbeagle [5] ($25)

All of these boards were examined and comparisons were made based on processor speed, memory, I/O capabilities, and compiler/development environments. A short comparison of these boards is given in Table 1.

Table 1: Linux PCB Comparison

Linux Board	Pro's	Con's
Raspberry Pi	• Low cost ($35) • Relatively large number of built-in device interface ports (e.g., USB, HDMI) • Large on-board RAM (>1 GB) • Pre-installed male headers (easy to wire) • High speed processor (1.5 GHz)	• Fewer I/O Pins (28) • Needs an external compiler • Needs H/W interface to run RS-232
BeagleBone Black	• Large number of I/O pins (72) • Pre-installed female headers (easy to wire) • Wide language compatibility • On-board programming environments • Multiple I/O processing units • High speed processor (1 GHz) with 2 additional 200 MHz programmable I/O processors	• Slightly higher cost ($55)
Pocketbeagle	• Large number of I/O pins (56) • Wide language compatibility • On-board programming environments • Multiple I/O processing units • Low cost ($25) • High speed processor (1 GHz) with 2 additional 200 MHz programmable I/O processors	• No pre-installed headers

One detail of note in this comparison is that the Beaglebone Black and Pocketbeagle have identical processing capabilities and differ only in their I/O capabilities, form factor, and, most noticeably, in cost. The hardware on each of these Beaglebone boards is a 32-bit, 1 GHz ARM (Cortex-A8) processor, and includes a socket for a microSD card (flash,) 512 MB of ram and an

on-board floating point accelerator. The Beaglebone boards also contain a pair of 200 MHz, 32-bit "programmable real-time units" (PRUs), which can gather data from all of the board's I/O pins simultaneous to any processes running on the Cortex-A8 processor.

On the Pocketbeagle, the microSD card also contains the operating system and long-term memory for the device. Conveniently, this means that, if there is ever an issue with the board (e.g., a student places the board, unprotected, in an area where it is at risk for ESD), the card can be removed from the damaged board, and placed into a new board, and the student can immediately proceed from where they left off.

As a result of this, if unintentional, safety feature, the lower cost of the board, its compatibility with Python, and large I/O capabilities of the Beaglebone products, the Pocketbeagle was selected for use in the present course.

PocketBeagle Board / Development Environment

The selected controller, the PocketBeagle, has a large assortment of I/O pins, including support for UART/SPI/I2C/PWM outputs, analog to digital inputs, and digital I/O. The board itself is pictured below in Figure 1.

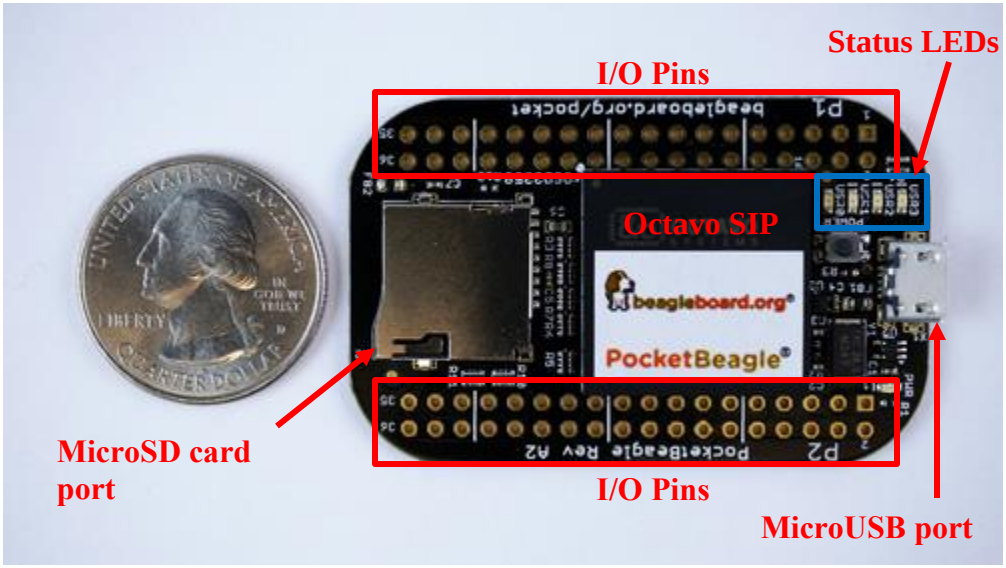

Figure 1: The PocketBeagle

The board's OS and on-board IDE can be accessed through via the microUSB port. While the primary interface to the board is this port, the board can be powered by battery as well.

The board's on-board hardware includes the following:

- Octavo Systems OSD3358 SIP, including:
  - 1 GHz ARM Cortex-A8 processor
  - 2x 32-bit 200-MHz programmable real-time units (PRUs)
  - 12MB DDR3 RAM integrated
  - 3D accelerator

    - Integrated power management
- 72 expansion pin headers, including eight 12-bit ADC inputs and 44 digital I/O ports
- MicroUSB port
- MicroSD port

A block diagram for the SIP is given in Figure 2.

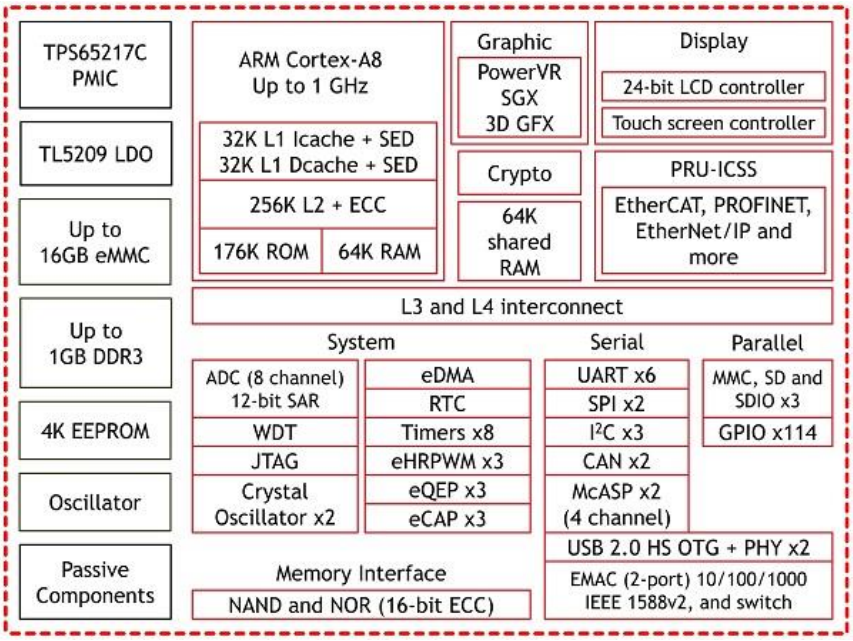

Figure 2: Octavo OSD335x C-SIP block diagram

The board runs a Linux kernel, enabling it to compile a variety of programming languages for interaction with the board. In this course, the PocketBeagle will be run under Debian v9.5 and programming will take place in Python v2.7. A more complete listing of the electrical specifications and hardware details is given in the System Reference Manual [6].

The PocketBeagle enables the user to run programs on up to three separate microprocessors: the main processing unit (32-bit, 1 GHz) and a pair of 32-bit, 200 MHz programmable real-time units (PRU). While the main processing unit can be used to compile and run programs and runs the board's (Linux) operating system, the PRU's do not have an operating system. Applications created for the PRU's have access to the I/O pins and shared memory (to communicate with Linux based programs). This arrangement enables the PRU's to run dedicated applications, processing data in real-time and turning the results over to the main processor. However, for this guide, the usage of the PRU's will be ignored.

The recommended Debian operation system offers I/O support to a number of programming languages, including the following:

- Python (Python v2.7 is native to Debian v9.5)
- BoneScript (an extension of Java Script)
- C application (direct hardware access)

- C application (via operating system file access calls)
- Java

The instrumentation class will utilize the on-board Cloud9 IDE to develop codes and interact with the Debian OS. This IDE can be accessed from the main menu, displayed after connecting the Pocketbeagle and browsing to 192.168.7.2, or by browsing directly to 192.168.7.2:3000. The IDE is shown in Figure 3 with the "Cloud9 Day" theme, selected from the "View" menu under "Themes".

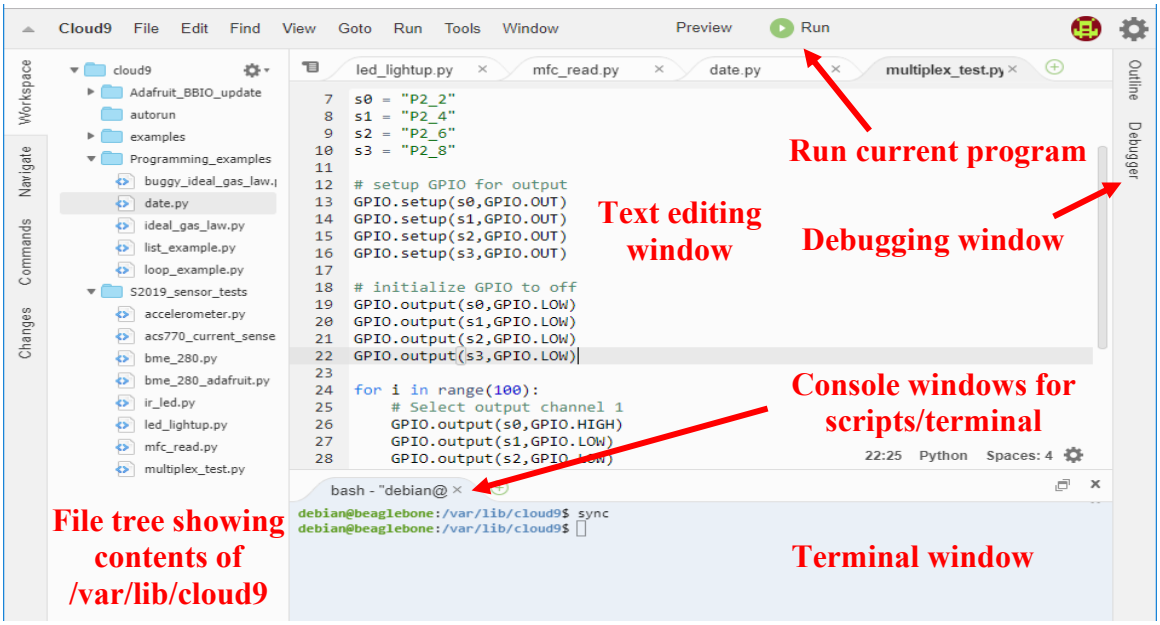

Figure 3: Cloud 9 IDE

This IDE provides both a text editor as well as terminal access to the OS, enabling the use of bash commands and allowing access to typical Linux programs (e.g., VIM, nano, etc.).

Lab Projects:

As the General Engineering program began at Saint Francis University during the 2018-2019 academic year, the instrumentation class has only been run once at the time of writing. In the first iteration of the class, there were a variety of class projects, three of which will be described in this paper:

1. Edge detection using IR photodiodes (lab)
2. Temperature measurement using an NTC thermistor (lab)
3. Fuel cell voltage/temperature monitoring (term project)

A description of each project is given below. Students were solely responsible for software development in each case, though each had to reproduce the given circuit layouts in order to debug their codes.

Edge Detection

In this application, a matched IR emitter/detector (Sparkfun SEN-00241) will be used as an optical switch. Both emitter (marked with a yellow dot) and detector (marked with a red dot) are infrared LEDs operating at a wavelength 940 nm and are shown in Figure 4.

Figure 4: Infrared emitter/detector (Sparkfun SEN-00241)

As is typical with LEDs, the shorter wire lead of each LED is the cathode, and the longer leg is the anode. When a voltage is applied across the emitter, it will begin producing IR radiation. The receiver, upon detecting this radiation, will close the circuit and allow current to pass through it. If the receiver does not "see" the emitter, it will remain open. Hence, by using a circuit similar to Figure 5 and monitoring the voltage between the receiver and ground, it is possible to identify when the receiver picks up the emitter.

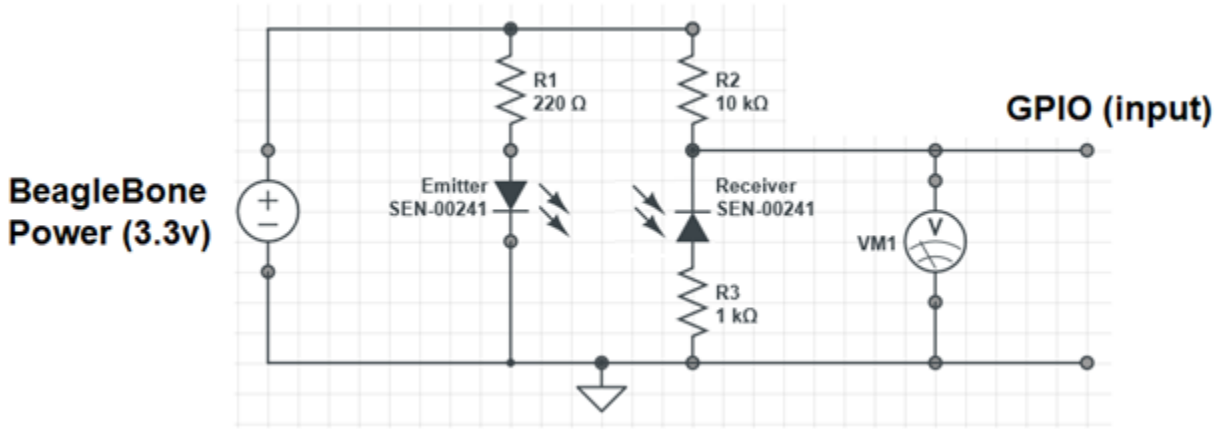

Figure 5: Emitter/Receiver circuit

In short, when the receiver side of the circuit is open, no current will be flowing through the receiving diode and the voltmeter should read approximately 3.3V. When the circuit is closed, the voltmeter reads the potential drop over R3, which is a comparatively small number, since R3 is small when compared to R1.

For the circuit to be closed, the receiver must "see" the IR radiation produced by the emitter. While the two diodes can be directly aligned with each other, providing a direct path for the IR "beam" to travel on, one can position the diodes next to each other and provide a reflective surface to "bounce" the beam from one to the other. In the latter case, the maximum distance from emitter to object appears to be 1 inch.

This sort of switch was considered for application as an optical propeller tachometer. In this

case, either method of edge detection can be used, although the reflective mode of operation may be desired due to possible changes in blade coning angle. In either case, the parameter of interest will be the change in voltage measured in Figure, which will change whenever the blade passing through the IR beam (the "sensing area"). For a tachometer based on reflection, it is anticipated that the voltage seen by the GPIO port will approximate the curve shown in Figure 6.

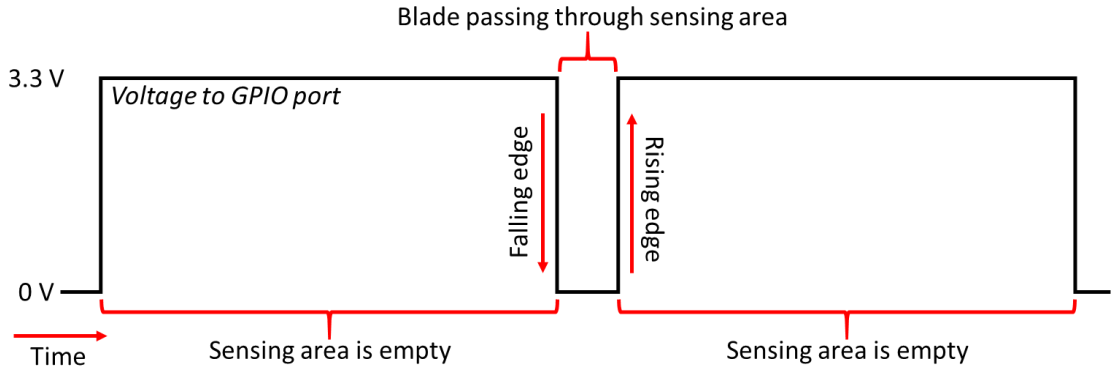

Figure 6: Reflective IR tachometer voltage output

In order to count blades only once/passing, it is desirable to count the falling or rising edges of the voltage input as opposed to whether or not the GPIO port voltage is simply high or low. The source code that enables event detection is shown below. The code uses the "add_event_detect" method, which can be set to trigger when the voltage to the GPIO port increases (a "rising edge") or decreases (a "falling edge") as a blade leaves or enters the sensing area. Running the "event_detected" method inside of a loop allows for continual monitoring of this event:

To determine the speed of the blade (in rotations per second), students used Python's time library to determine elapsed time.

Temperature Measurement

To measure temperature, the students will use a waterproof DROK B3950 temperature probe (rated for -25°C to 125°C), which utilizes an NTC thermistor for thermal sensing. Thermistors, a form of resistive temperature detector (RTD), are especially made to have a high sensitivity to temperature, as shown by the resistance/temperature lookup table for a B3950 thermistor in Appendix 6.1. This relationship, while nonlinear, can be modeled by using the Steinhart-Hart equation, which is given in Equation 1. In this expression, T is the measured temperature, R is the resistance of the thermistor, and A, B, and C are constants.

$$\frac{1}{T} = A + B\ln(R) + C\,[\ln(R)]^3 \qquad (1)$$

The constants in this expression can be determined via calibration over a specific temperature range. It is worth noting that selection of the calibration range is important as 3 calibrations points will be necessary: one at the upper end of the measurement range, one at the lower end of the measurement range, and one in the middle. The placement of the middle point determines the regions where the calibration curve will most accurately represent the correct temperature, as

is illustrated in Figure 7.

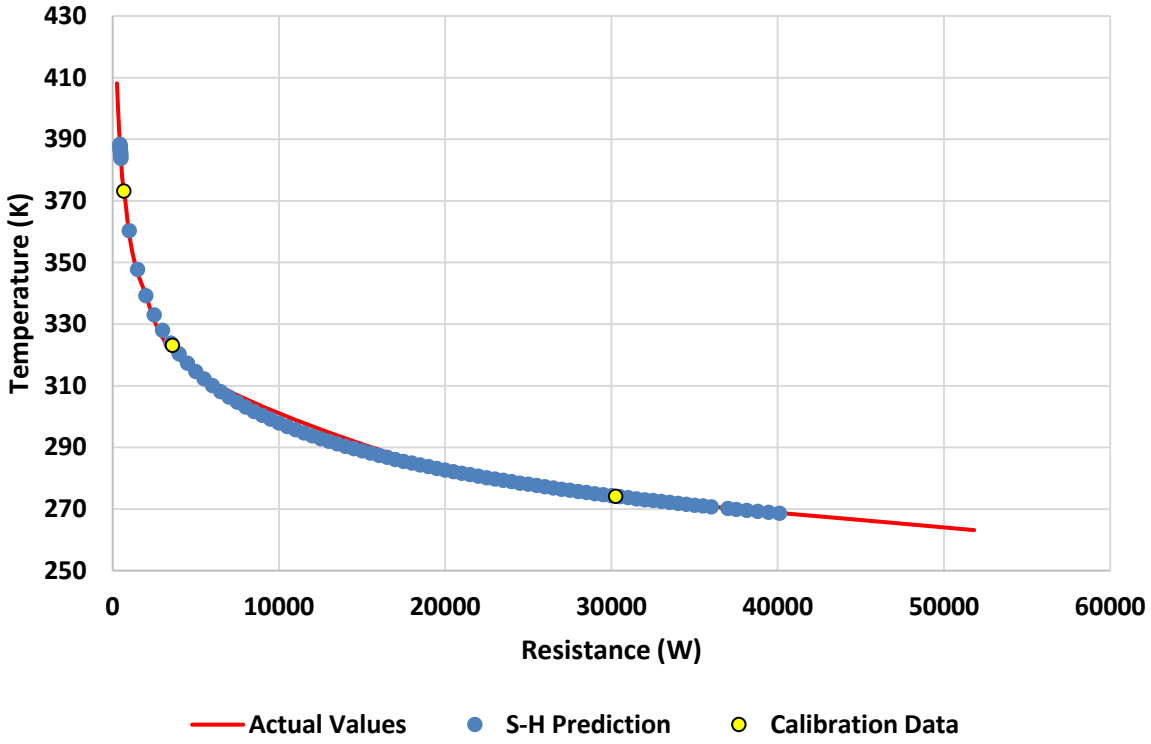

Figure 7: NTC Thermistor Modeling

Direct computation of the constants in the Steinhart-Hart equation can be accomplished as follows, using three calibration points with resistances R_n and temperatures T_n. Given this process, the only required input is resistance, which can be measured indirectly via voltage. This can be assessed by an appropriately sized voltage divider, as shown in Figure 8.

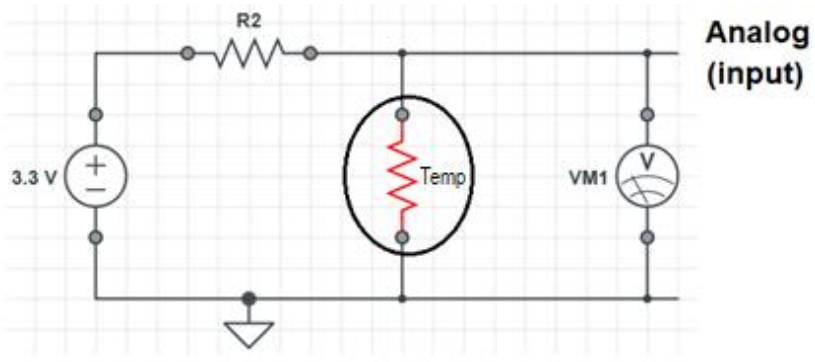

Figure 8: Voltage divider with Thermistor

The size of the resistor *R2*, must be appropriately selected to both limit current flow and provide a proper voltage input (0 to 1.8 V) to the PocketBeagle based on the anticipated operational domain of the thermistor.

The method for obtaining resistance from thermistor voltage can be derived from Ohm's Law

and the voltage input to the circuit, V_{input}, as shown in Equation 4.

$$V_{TEMP} = I \cdot R_{TEMP} \tag{2}$$

$$V_{input} = I \cdot (R_2 + R_{TEMP}) \tag{3}$$

$$V_{TEMP} = V_{input} \frac{R_{TEMP}}{R_2 + R_{TEMP}} \tag{4}$$

Based on this expression, one can develop an expression for the resistance of the thermistor given a fixed resistor, R_2, a known voltage input, V_{input}, and a sensed voltage drop across the thermistor, V_{TEMP}. Any resistor over 1000 Ω should prove suitable for R_2.

Fuel Cell Voltage/Temperature Monitoring

The challenge of this particular project was to combine past knowledge and apply it to a real project. The project itself was provided by a fellow faculty member who was interested in a replacing an aging voltage measurement system. This system was used to measure the voltage output from a set of up to 40 microbial fuel cells (MFCs).

MFCs are similar to fuel cells or batteries in that they have an anode where oxidation is taking place and a cathode where reduction is taking place. Between the anode and the cathode, a resistor has been connected, across which can be measured the voltage of the entire cell. This measurement is currently made by a multimeter that reads the voltage for an MFC across the resistor using two alligator clips, one attached to the cathode, and the other attached to the anode. The alligator clips are at the end of long wires that connect to headers in the circuit board within the multimeter. Data is collected every 20 – 30 minutes, and the graph we produce from this shows voltage vs. time (Figure 9). Each peak represents a single "batch" in which the food oxidized by the bacteria in the MFC is consumed.

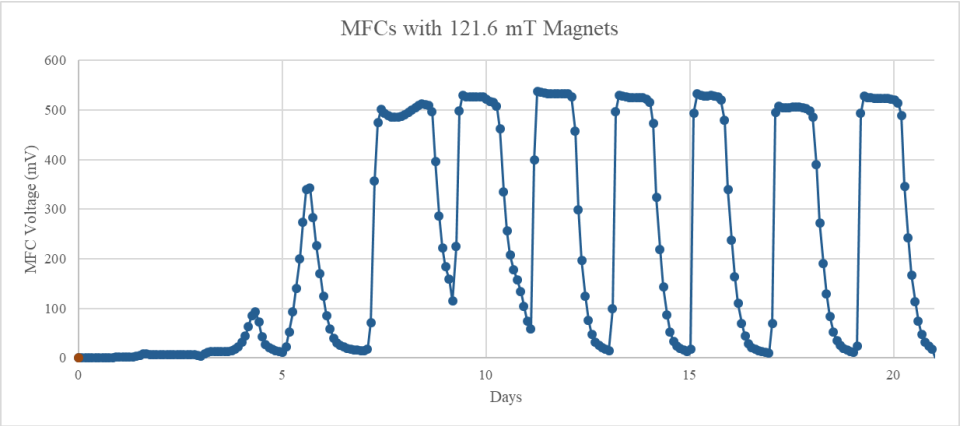

Figure 9. MFC sample data of voltage vs. time.

The typical MFC is read across a 1000-ohm resistor and yields a peak voltage of approximately 500-550 mV. However, certain tests require variation of the resistance, and thus variation in the voltage response, sometimes as low as 5-10 mV using a 10-ohm resistor. The MFCs can be kept

at room temperature or at 30°C, depending on the experiment, but the wires can lead out of the incubator to the data collector, which can be maintained at room temperature.

In order to accomplish this project, the students' code had to meet the following requirements:

- Record voltage information according to the following schedule:
  - For 40 MFCs: > 0.00333 Hz (one set of 40 MFC readings/5 min)
  - For at least one additional MFC: > 4 Hz
- Record the temperature in the oven at the time of any voltage measurement
- Data from each channel must be recorded and tracked separately, including time and voltage for all measurements
  - Time should include an accurate log of the day, month, year, and second of each measurement
- Data from all channels should be written to a single document at least every 20 minutes

While the PocketBeagle does have a limited number of analog input pins, it is possible to extend this by using a multiplexer to meet the needs of the project. In short, a user can select a single channel (in the case of the device listed in Figure 10), out of 16 possible input channels on the device and then connect that to a single signal pin. Any input to that channel will be available at the signal pin, and, conversely, any input to the signal pin will be routed to the selected channel.

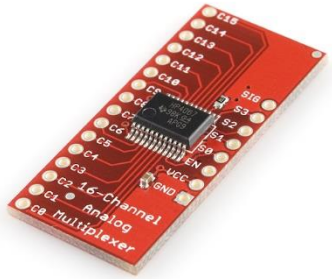

Figure 10: 16 channel analog/digital multiplexer (Sparkfun: CD74HC4067)

The multiplexer utilized in this class is a 16 channel unit that can be powered from a 3.3V or 5V source, although the 5V source onboard the PocketBeagle was utilized for this application, as shown in Figure 11. Please note that selection of a 5V power source still allows for 3.3V logic typical of the PocketBeagle.

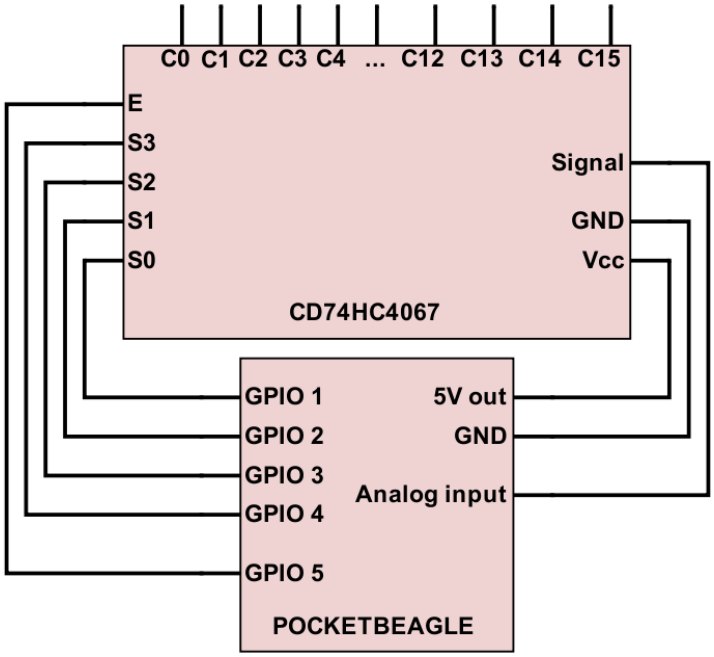

Figure 11: Multiplexer wiring diagram (for analog input)

Multiplexer channel selection is performed by manipulation of the PocketBeagle's GPIO ports to input a binary value to the pins labeled S0-S3 on the multiplexer, and the multiplexer itself is enabled via the "E" pin. To select the proper channel, pins S0-S3 on the multiplexer correspond to binary place values 1, 2, 4, and 8, respectively. Entering the binary equivalent of the desired base 10 channel number using these pins will select the desired channel.

Once a channel is selected, it is recommended to allow for a 50 ms delay before read/write operations. By using a logical AND operation on the modulo of a desired channel number, the program can output the binary value of the desired (multiplexer) value. Once read, the challenge for the students was to collate recorded information with a particular MFC and save it for future retrieval.

ABET Outcomes

Once completed, each of the final lab projects could be used as instruments for a variety of ABET [10] student outcomes. While this course is presently used to evaluate outcomes 1 & 6 in Saint Francis University's general engineering program, the microcontroller and lab activities described can easily be utilized in projects applying to other outcomes as well. Some possibilities for this integration are discussed in

Table 2.

Table 2: ABET Outcomes and Course Projects

Outcome	Integration with ENGR335
1. An ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics	Description of theory behind various sensing methods; solution to data acquisition problems (temperature measurement lab)
2. An ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors	Use of a real-world data acquisition problem as a course project (MFC voltage sensing project)
3. An ability to communicate effectively with a range of audiences	Presentation of course project to the client (MFC voltage sensing project)
4. An ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts	Selection and justification of components for any project, weighing the triple bottom line of sustainability, economic, and social impacts
5. An ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives	Collaborate with others in the accomplishment of lab activities, but specifically the term project
6. An ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions	Interpretation of acquired data, identification of trends, constraints, abnormalities, etc. (all lab experiments)
7. An ability to acquire and apply new knowledge as needed, using appropriate learning strategies.	Design of a data acquisition project outside students' focus area (MFC voltage sensing project)

Assessment

A survey was provided out to the students upon completing their final lab project. The students were asked to compare and rate their experience of the course. Ratings were given on a 1-5 scale (e.g., lowest, moderately low, neutral, moderately high, highest) according to both the student's experience of the class with regard to general educational objectives and ABET outcomes. Feedback was gathered via an IDEA survey, an anonymizing, online feedback tool created by Campus Labs.

Feedback from the course (all three students responding) was generally positive, indicating student progress in the understanding and application of course material. Specific feedback from

the IDEA survey pertinent to this is shown in Table 3. In this chart, a response of 1 corresponds to no apparent progress on a particular item and 5 corresponds to exceptional progress on an item.

Table 3: Student progress on relevant objectives

Items	Average Rating (1-5)	Std. Deviation
Gaining a basic understanding of the subject (e.g., factual knowledge, methods, principles, generalizations, theories)	4.00	0.00
Learning to apply course material (to improve thinking, problem solving, and decisions)	4.33	0.47
Developing specific skills, competencies, and points of view needed by professionals in the field most closely related to this course	4.00	0.00
Learning appropriate methods for collecting, analyzing, and interpreting numerical information	4.33	0.47

The most common response collected in the survey was to add a pre-requisite course in programming. In this first iteration of the course, no programming courses were listed as pre-requisites for the class, requiring that the first 1.5 months of the course be utilized to teach Python programming. This was found to be undesirable by both students and the instructor, since less material could be covered in class and the students were less familiar with the language from the outset.

With regard to the ABET Criterion 3 outcomes, the students rated their progress in this course with respect to the 7 outcomes, as shown in Table 4. As before, a response of 1 corresponds to no apparent progress on a particular item and 5 corresponds to exceptional progress on an item. Given the course's content, it is presently used to assess outcomes 1 and 6 in the junior year of general engineering program at Saint Francis University.

Table 4: Progress on ABET Criterion 3 Outcomes

Outcome	Average Rating (1-5)	Std. Deviation
1	***4.11***	***0.85***
2	4.67	0.47
3	2.67	1.70
4	2.33	1.73
5	1.33	0.47
6	***4.17***	***0.86***
7	3.33	1.25

The very high and consistent progression on ABET Outcome 2 reflects the course's focus on applications. While ENGR335 was not conceived as a design course, the project-based learning

scheme and final design project have successfully transmitted a design-centric mentality to the students. This was specifically targeted by the selection of a final project that served the need of an in-progress research project and was designed by that project's principal investigator. Furthermore, as evidenced by strong scores in Outcomes 1 and 6, this was accomplished while maintaining a focus on fundamentals of the problems at hand and enabling the students to develop their engineering intuition by interaction with real-world systems.

One of the students in the class, having significant experience with other Linux-based microcontrollers, provided a comparison of the Pocketbeagle and the Raspberry Pi. This was obtained via an informal survey, requesting his input as a student and user of the two devices. Results of this survey are given in Table 5.

.

Table 5: Comparative Survey of Student Experience

Question	Response
Compared to to the Raspberry Pi, how easy was the BeagleBone hardware to learn and understand? Rate each on a scale from 1-5 (1 – most difficult, 5 – easiest).	The beaglebone and the raspberry pi were similar in difficulty. The pocket beagle is a simpler system than the full raspberry pi, but does not have as much documentation and forum support as the pi does. **Beagle: 4/5 Pi: 4/5**
Compared to your IDE of choice (please note in your answer) for the Raspberry Pi, how easy was the Cloud9 development IDE to bring up, understand, and develop code in? Rate Cloud9 on a scale from 1-5 (1 – most difficult, 5 – easiest).	The Cloud9 IDE is very easy to use and understand. Compared to IDEs like Netbeans and InteliJ, there are some debugging and analysis features missing, but the easy access and ease of use offset these downsides. Raspberry Pi programming is not nearly as straight forward. **Comparative quality of Cloud9: 5/5**
Compared to other microcontrollers, was the Pocketbeagle board reliable Rate each on a scale from 1-5 (1 – unreliable, 5 – most reliable).	On a whole, the beagle performed well. However, I experienced recurring issues where the pocket beagle would not fully boot, and it would take several attempts to get the OS to boot. **Reliability: 3.5/5**
Was the Pocketbeagle introductory material provided helpful to the course? Rate the material on a scale from 1-5 (1 – not helpful, 5 – helpful).	4. Yes, the introductory material was helpful for the course. It helped me get acquainted with the hardware and the custom OS on the beagle. **Material quality: 5/5**
Do you have an interest getting a job/career that uses embedded linux? Rate your interest on a scale from 1-5 (1 – not at all, 5 – definitely).	Yes, I am interested in embedded systems as a potential career path. **Career interest: 3.5/5**

With respect to the career aspirations of the student in question, please note that the student went on to intern at NASA Houston the next Spring, developing embedded systems to support a VR training environment.

In addition to the feedback noted in Table 5, the student also commented on the necessity of a pre-requisite programming course prior to ENGR335, and suggested that students taking ENGR335 should also take electrical engineering course simultaneously (this is the arrangement in the present paradigm). The student also noted a great interest in the further development of similar classes, and, since taking ENGR335, has taken a control systems class that utilized the Pocketbeagle to host a PID control system.

While the student surveyed noted that typical labs took an average of 5 hours to complete, it is worth noting that this is not the average for a typical student in the class. As such, the pair of 50 minute class periods provided for lectures and labs proved much too short for any meaningful lab experience. In its next offering, ENGR335 will be offered as a 1-credit class, but with a single 1 hr 50 min class period each week to enable greater depth for in-class lab experiences. Furthermore, as the provided introductory material was incomplete at the time of the class, it is anticipated that the completed class manual will allow for more rapid familiarization with the Pocketbeagle as well as the various sensors utilized in the course.

Summary

There are many embedded environments that can serve as a platform for measurement and control applications. The PocketBeagle with its embedded Linux operating system performs quite well and supports all of the basic I/O that is needed for typical mechanical engineering test and measurements. This includes digital I/O, analog input PWM output, I2C, SPI and RS-232 communications, as ell as ethernet and USB support. The PocketBeagle's I/O can be further enhanced by using digital and analog multiplexor's.

The PocketBeagle also supports a number of programming languages and this paper describes how Python could be used as a basis for a mechanical engineering course.

What makes the PocketBeagle more valuable as a platform is its operation within an academic setting. In such a setting, a development IDE that is simple to use, easy to install and low-cost makes it an ideal solution. The PocketBeagle does not need any specialized wiring or electrical circuits in connecting the board to a PC. The development software (Cloud 9) is already installed on the board along with the drivers to connect with a PC.

Two of the main comments made by the students is (1) that some of the lab required a significant amount of time to complete and (2) the suggestion to require a Python class prior to this. Knowing Python ahead of time would also help to decrease the time out-of-class that was needed to complete the lab assignments.

Bibliography

[1] Web Site: https://beagleboard.org/black/
[2] Web Site: https://www.raspberrypi.org/products/

[3] Web Site: http://www.glomationinc.com/products/index.php?n=SBC.GESBC-9G20u
[4] Web Site: https://www.sparkfun.com/products/12857
[5] Web Site: https://www.arrow.com/en/products/pocketbeagle/beagleboardorg
[6] Web Site: https://github.com/beagleboard/pocketbeagle/wiki/System-Reference-Manual