AC 2011-2234: THE CHIRPS PROTOTYPING SYSTEM

Gary Spivey, George Fox University

Gary Spivey received his BS in electrical engineering from the University of Arizona and his MS and PhD degrees from University of Maryland at College Park.

He has served as an electronic engineer with the National Security Agency, chiefly as a special-purpose-computer and application specific integrated circuit (ASIC) designer. His current research interests include FPGA system development and modular embedded systems.

The Chirps Prototyping System

Abstract

Oregon State University has been a pioneer in developing a "Platform for Learning" using their TekBots platform as a fundamental part of their electrical and computer engineering curriculum. At George Fox University, we fundamentally affirm this concept of a "Platform for Learning," but we additionally desire a "Platform for Prototyping." By "Platform for Prototyping," we mean a platform that will enable our engineering students to create significant engineering projects as part of a myriad of service-learning projects, student research, course projects, and the senior capstone experience. To be effective across our curriculum, this system must not only be usable by mechanical, electrical and computer engineers, but by engineering students at the end of their first year in the engineering program.

As it is difficult to conceive of a significant engineering application that does not have some form of embedded control system, it is almost imperative that these students obtain the ability to understand and control some form of an embedded control system early in the curriculum. This presents some challenges. Many embedded processing systems make use of one microcontroller controlling a variety of sensors and actuators, requiring that one microcontroller program be written to control every detail of the embedded system. Even simple embedded systems can require a multitude of tiny details including modulation for multiple infrared sensors, pulse-width modulation control for various dc and servo motors, and interfaces to various components such as LCD displays and wireless interfaces. This level of programming sophistication is generally reserved for upper-division ECE courses where interrupts and timers are discussed in great detail.

To address these issues, we have developed a distributed embedded processing system called "Chirps." Chirps are a suite of boards that can communicate with one another using short command bursts, or "*chirps*." Rather than having a central controller that needs to manage pulse-width-modulation and encoder processing for a variety of motors, a Chirp system will contain a Chirps motor controller board that can be accessed using simple commands such as "Move Forward 100 ticks." This Chirp abstraction will provide users the ability to rapidly assemble and control larger systems (such as robots) from Chirp modules and easily control them using "chirps." Although detailed functions are moved to individual control boards, a central controller must still be provided to control the system and send and receive "chirps." For this purpose, we have enhanced the very popular open-source suite of hardware and software provided by the Arduino project. The Chirps controllers are built on the foundation of the Arduino Diecimila board, with a power and communication interface added to facilitate the "chirp" system.

In the first-year engineering sequence, students are taught how to program the basic controller and make calls to the various Chirps boards. Upper-division ECE students develop new Chirps boards as part of the microprocessor course. These boards are targeted toward the needs of the various projects being developed throughout the curriculum. Using the Chirps system, Y University engineering can rapidly prototype and control a variety of significant engineering systems.

Introduction

George Fox University (GFU) engineering students have been very successful in developing a myriad of substantial electromechanical systems in the capstone senior design experience[1]. Students have obtained a significant amount of experience and confidence through designing and building these systems. It has long been a desire of the department to see our students participate in these types of design/build experiences throughout the curriculum. Oregon State University (OSU) has been a pioneer in developing a "Platform for Learning[2]" using their TekBots platform[3]. The TekBots platform is an electronic robot vehicle that is built by every student in the electrical engineering program at Oregon State University. The program sponsors state that "using a common platform throughout the curriculum helps to integrate the material from seemingly disjointed courses[4]." While the Oregon State University TekBots program provides a model of a Platform for Learning, it is not necessarily a perfect platform for other Universities. The sponsors recognize that "a very large number of equally useful platforms for learning could be conceived that provide the aforementioned benefits … in the future, the platform could become a boat, an airplane, or a submarine[5]."

Every GFU engineering student will participate in a two-year extensive engineering service program and many students participate in undergraduate research projects. Thus, GFU students are required to build a significant number of electromechanical systems. As GFU is a teaching institution without engineering graduate students, there are some significant challenges to building these systems – namely, the difficulty in obtaining and designing control systems for the electromechanical designs. It is, in fact, difficult to conceive of a significant engineering application that does not have some form of embedded control system[6*]. Therefore, we set out to find or develop a modular system that will enable students to adapt elements to other engineering problems. It was our goal that students who are equipped to control this platform would also be able to control a myriad of other engineering systems.

Typical off-the-shelf control systems can cost several hundred to several thousand dollars. Many require a host computer that severely limits mobility. These systems can be designed and fabricated much more economically in-house, but, at GFU, the skills to do so are generally taught in the spring of the junior year to electrical engineering students. This limits the complexity of student-driven design/build projects at earlier stages in the curriculum, where students will be required to participate in significant design/build projects before having the requisite skills to design the systems that control them.

To address this issue, we have long been working to develop a system that would both mimic the OSU model of the "Platform for Learning" and also enable students to design and build system controllers using skills obtained in the freshman year and enhanced throughout the

[*] The Milwaukee School of Engineering (MSOE) is a very large engineering school with separate tracks for computer engineering and electrical engineering. At the 2005 American Society of Engineering Educator's Conference, presenters from MSOE stated that *every* electrical engineering senior design project for the past several years included a microcontroller.

curriculum. Such a system will be both a "Platform for Learning" and a "Platform for Prototyping". That is, George Fox University engineering students will not only be able to understand engineering concepts, but will also be equipped to utilize those concepts in rapidly developing real systems for service or research projects.

History

The year-long freshman engineering sequence at GFU had been centered on two primary projects:

- the design and manufacture of an oscillating steam engine (commonly referred to as a "Wobbler")[7], and
- the operation of an autonomous robotic arm through the use of a modular Matlab (computer) program.

These projects serve as the basis for the integrative format of the course, and had proven to provide a highly effective hands-on platform for the instruction and experience in several applicable design skills and tools[8]. The first project (Wobbler) includes elements of engineering analysis, mechanical design, virtual prototyping (solid modeling), and fabrication / machining skills. The second (robotic arm) includes elements of robotics, real-time control, machine vision, artificial intelligence (programming skills), and instrumentation. However, the robotic arm required MATLAB to run on a host computer. While many computer skills were being taught, embedded control was not among them.

As the freshman course was already using MATLAB, we desired to leverage the MATLAB Embedded target software and decided that the viable controller option for us was the Freescale HC12[9]. This board was completed and shown to work well; however, seamless operation was not possible as the current MATLAB embedded target did not support the latest HC12 compilers. In order to allow freshmen engineering students to do a seamless integration from MATLAB to a microprocessor, we began working with MATLAB's Real-Time Embedded Coder to create generic 8-bit microprocessor code for mapping onto an Atmel AVR microprocessor on a redesigned controller board. As this project began to unfold, the complexity-level of the student interface began to increase. In order to utilize the Real-Time Embedded Coder, students would be required to interact with Simulink in addition to MATLAB. Separate processes would need to be developed to hide the processes of MATLAB to C code conversion as well as C to microprocessor object file creation and the downloading for the code from our system to the system hardware. Any errors in the process would be confusing for students as there would be so much going on "under the hood." We were also disappointed to discover that the MATLAB Real-Time Embedded Coder would not work with all MATLAB code, only a subset. This restriction could cause more confusion with students who must become *very* familiar with what MATLAB code will work. Any errors in this area would also appear "under-the-hood." While headed down this path, we became discouraged with the approach and realized that integrating MATLAB autonomously might require much more from the freshman engineers than we believed to be reasonable.

In June of 2007, MAKE© magazine published an article entitled "Arduino Fever" that detailed the introduction of a "cute, blue microcontroller that fits nicely in the palm of your hand, and the expanding community of developers who love and support it. [10]" The project was designed to be completely open-source – both hardware and software. "The original intention of the Arduino project was to see what would happen if community support were substituted for the corporate support that is usually required for electronics development." The result has been an explosion of community interest, a website (http://www.arduino.cc) that includes software, sample code, forums, hardware schematics, language references – in short, everything required for a hobbyist to develop a controller.

What we found initially intriguing was that the Arduino made use of the very same microcontroller that we were using (the Atmel AVR). More significant was the software-programming environment provided. Utilizing a very simple version of the 'C' programming language, the makers of Arduino put together a complete software integrated development environment (IDE) that enabled simplified programming and one-step download of the code to the Arduino hardware. The code framework consists of two main functions, *setup* and *loop* that programmers use to control the initial setup of the embedded system and the repeated control of the system. Also provided are simplified functions such as *digital_write*, *digital_read*, *analog_read*, and *delay* to allow users to control the timing of events on the pins of the microcontroller. While this system was not MATLAB-based, the programming environment was simple enough for high-school students to utilize successfully.

It occurred to us that using MATLAB as an initial introduction and direct controller to our system would be more than sufficient training to allow students to migrate to the Arduino system to accomplish *autonomous* control. Furthermore, with the increasing number of support boards available, students could leverage the Arduino community along with the GFU platform to create a wider range of engineering service and research projects.

Chirps Overview

As we began to utilize the Arduino platform, we discovered what we felt were two limitations in adopting the Arduino as our embedded controller – extensibility and abstraction.

Arduino boards extend the pins of the AVR microprocessor to headers that allow a daughter card (or "shields" as the Arduino community references them) to be attached for extensibility. There are a myriad of shields such as the Motor Shield and the XBee Shield (for wireless communication). These shields are wonderful in the event that an individual wants to control a motor or communicate wirelessly. However, what would an individual do if they wanted to control a motor *AND* communicate wirelessly? Liquidware has developed a "doublewide extender shield" that replicates the pins to another set of headers. However, this sees an obvious limit when a user might decide that they want a sensor-activated motor controller with wireless communication. Fundamentally, it was our desire to add a defined-communication mechanism to the Arduino platform that would allow a large number of boards to be connected in a system.

The Arduino community is large and helpful. There are a wide range of software libraries that allow functionality to be integrated into control programs. However, the embedded controller must find the libraries, and then must integrate them into the embedded control program on the Arduino. The size of these applications can rapidly fill the programming space of the controller. Furthermore, many of these libraries as well as accompanying shields require that the user understand the electronics of the device to be controlled. We felt that it was important to abstract the control of these devices so that system designers would be able to focus on the system, not the details of particular electronic interfaces.

To rectify these limitations, we have developed an augmented system based on the Arduino. This system is known as "Chirps." Chirps are a suite of boards that can communicate with one another using short command bursts, or "chirps." Rather than having a central controller that needs to manage pulse-width-modulation and encoder processing for a variety of motors, a Chirp system will utilize a motor controller board that can be accessed using simple commands such as "Move Forward 100 ticks." This Chirp abstraction will provide users the ability to rapidly assemble and control larger systems from Chirp modules and easily control them using "chirps." While detailed functions are moved to individual control boards, a central controller must still be provided to control the system and send and receive "chirps."

Chirps Hardware

The Chirps controllers are built on the foundation of the Arduino Diecimila board (and look like one to the computer), but a power and communication interface has been added to facilitate the "chirp" system. The Arduino Diecimila is built around the Atmel ATmega168 microcontroller (one chip in the Atmel AVR family). The board has a standard 6-pin In-System-Programming (ISP) header for programming using special hardware and software, but what really makes them simple and attractive is the integration of the hardware bootloader into the Arduino software IDE. The microcontroller bootloader is coded to examine the serial interface lines on reset. If there is no activity for one second, it proceeds to run its previously installed program. However, if there is activity, it will emulate the Atmel STK500 protocol for programming the part (which allows it to use the open-source avrdude programmer). These Arduino boards make us of an FTDI232R USB→UART converter chip (later boards have migrated to an Atmel ATmega8u2 for this interface) that allow simple connectivity between most any computer and an Arduino board. It was this core from the Arduino Deicemilia that was augmented into the Chirps controller board.

Chirps boards are interconnected using CAT-5 cables; however, they do not communicate using Ethernet protocols. The CAT-5 cables along with the RJ-45 connectors were selected because they are abundant, affordable, and easy to assemble, allowing system designers to place boards in a wide variety of locations relative to each other. The RJ-45 connectors are also robust and provide a locking connection – both features that were important for any motorized system. Each Chirps board contains two RJ-45 connectors to allow the "daisy-chaining" of a large number of Chirps boards. While the RJ-45 connectors are not as small as some others, we decided that the benefits of functionality outweighed the size limitations. Ultimately, the Chirps system is a prototyping system, and size is a minor consideration for a prototype.

Power is distributed unregulated throughout the system and is then regulated on each board. This method results in lower currents on the power wires and allows each board to have unique voltage values for various motors and logic functions. Generally speaking, we felt that Chirps systems should be able to drive up to 24V motors off of a standard power supply, so most Chirps boards utilize a switching 3.3V regulator that allows up to 36V input.

Communication is performed via the I^2C interface (Two-Wire interface on the Atmel AVR microcontrollers) on each board. One of the drawbacks of the I^2C specification is the capacitive limit of 400pF on the bus. This results in a limited range due to the dependence on passive pull-up resistors to provide the pull-up current. To address this problem, the Chirps system uses a current mirror in place of the pull-up resistors to improve pull-up performance. Use of excessive current in the current mirror resulted in reduced slew rate during switching 'low' while insufficient current resulted in reduced slew rate during switching 'high'. After testing, the optimal pull-up current appeared to be approximately 2mA. Whave tested the current drivers up to capacitances of 20nF. More practically, the Chirps system has communicated through 25 feet of CAT-5 cable with no appreciable degradation of signal quality.

The basic I^2C specification allows for the use of a 7-bit address to identify each unique device. As all devices to be communicating on the I^2C bus will be Chirps modules, the Phillips official addresses could be ignored and the entire address space used for Chirps module addressing. The first and last 8 address in the address space were left reserved as per I2C specifications to allow the use of global broadcasts and the future possibility of 10-bit addressing. Chirps boards are uniquely addressed through a combination of board type and the setting on a 4-pin DIP switch on each board.

Chirps Software

Only the Chirps controllers utilize the Arduino model. All other Chirps boards are designed to be programmed by the designer, and then interact with the controller board using the Chirps communication interface. The Chirps communication protocol implements the multi-master I^2C protocol. This allows any board to communicate to any other board whenever required. Boards are allowed to send or request data from other boards. If a particular board experiences difficulties and resets, it can request a re-initialization of its values from the controller. The controller may also send command packets (either to one board or as a broadcast) that indicate a start or stop event. Without accounting for multi-master collisions, the Chirps bus can maintain 2-3 packets per millisecond.

Each board provides its own API that sits on top of the Chirps standard communication protocol stack.

The Chirps Protocol Stack is made up of the following levels:

1. I^2C Protocol – Provides access to the physical medium. Communications at this level consist of data packets containing a device address, a read-write bit, and an unknown number of data bytes.
2. Chirps API – Defines device addresses based on device types and adds packet length and CRC bytes to each data packet. It also defines several possible packet types.
3. Module Attribute Control – These are a group of objects, one for each board, with a varying number of attributes that can be modified by the User Application.
4. User Application – The program written by the end user, currently compiled using the Arduino IDE. Each Chirps board provides an API for the application layer.

The Chirps API makes use of several different packets including:

- Data Packets – Write command packet that contains data to be written and the location where it will be written.
- Request for Initialization Packet – A command packet that indicates the board has reset and is requesting to be reinitialized.
- Initialization Packet – A data packet with initialization parameters.
- Start Packet – A command packet indicating that a start event is occurring.
- Stop Packet – A command packet indicating that a stop event is occurring.

Firmware on Chirps module utilize the following Chirps API functions:

- chirp_sendTo – sends a packet to a board.
- chirp_requestFrom – requests a packet from a board.
- chirp_requestInit – A request from a board to a controller indicating that it has reset and needs to be reinitialized.
- chirp_sendStart – a synchronous packet that can be broadcast to all boards in the system indicating a start event.
- chirps_sendStop – a synchronous packet that can be broadcast to all boards in the system indicating a stop event.
- chirp_attachSlaveReceive – attach a function to be called when a master device has performed a sendTo to this device.
- chirp_attachSlaveReply – attach a function to be called when a master device has performed a requestFrom to this device.
- chirp_attachInitReceive – attach a function to be called when an initialization request is being answered.
- chirp_attachOnChirpStart – attach a function to call when a start command is received
- chirp_attachOnChirpStop – attach a function to call when a stop command is received.

The core of every data packet is an Attribute Address and its associate data consisting of 1 to 27 bytes of information. Table 1 identifies the packet structure of a Chirps packet.

Table 1: Chirps Packet Structure

Information:	**Number of Bytes:**
Destination Device Address	1
Packet Length	1
Source Device Address	1
Attribute Address	1
Data Byte(s)	1–27
CRC Byte	1
Total:	***6–32***

Chirps Modules

Several Chirp modules were developed as the initial implementation of the Chirps system. These include the:

1. Raven – The Chirps controller based on the Arduino Diecimila.
2. Macaw – A wireless controller similar to the Raven where the USB has been replaced with an XBee wireless chip.
3. Parrot – provides a USB-Wireless connection that allows a computer to talk to the Macaw (or any other XBee).
4. Roadrunner – controls 2 DC motors with encoder inputs.
5. Woodpecker – controls 6 DC motors or 3 stepper motors.
6. Hummingbird – controls 18 Servo-motors.
7. Owl – manages 8 infrared photo-transmitter/detector pairs.

As an example of the User Application API, Table 2 and Table 3 provide the API calls for the Roadrunner and Owl boards.

Table 2: Roadrunner API

Function	**Purpose**
Roadrunner *NAME*(*address*)	Set the address of the Roadrunner.
NAME.setMotor1(*speed*, *dir*)	Turn on motor 1 and set its speed and direction.
NAME.stopMotor1( )	Turn off motor 1.
NAME.invertMotor1( )	Switch which direction is considered *forward* by motor 1.
NAME.setMotor2(*speed*, *dir*)	Turn on motor 2 and set its speed and direction.
NAME.stopMotor2( )	Turn off motor 2.
NAME.invertMotor2( )	Switch which direction is considered *forward* by motor 2.
NAME.setMotors(*Motor1_speed*, *Motor1_dir*, *Motor2_speed*, *Motor2_dir*)	Set the speed and direction of each motor with one call.
NAME.stopMotors( ):	Stops both motors.

Table 3: Owl API

Function	**Purpose**
Owl *NAME*(*address*)	Set the address of the Owl..
NAME.getSensors( )	Returns the digital values of all 8 sensors.
NAME.getSensor(*sensor*)	Returns the digital value of a specific sensor.
NAME.getSensorValue(*sensor*)	Returns the analog value of a specific sensor.
NAME.setSensitivity(se*nsitivity*)	Set the sensitivity gain of all sensors.
NAME.setSensitivity(*sensitivity*, *sensor*)	Set the sensitivity gain of a specific sensor.
NAME.setThreshold(threshold)	Set the threshold of all sensors.
NAME.setThreshold(threshold, sensor)	Set the threshold of a specific sensor.
NAME.setUpdateMode(interval, changes_only,analog_mask)	Set the interval for automatic updates as well as the content of the updates.

To provide a training system for the Chirps modules we developed a robot that is controlled by Chirps boards. This robot is called the PUMA (see Figure 1) and it allows for hands on, exciting, and practical use of the Chirps boards by engineering students in the second semester of the freshman engineering experience. Each PUMA contains a Macaw for control, and a Roadrunner and Owl board for the motors and lights respectively. To control a PUMA, a user would make use of the Arduino IDE loaded with the Chirps libraries. Then by simply making the appropriate API calls, a user can use sensor input to control the motion of an autonomous robot.

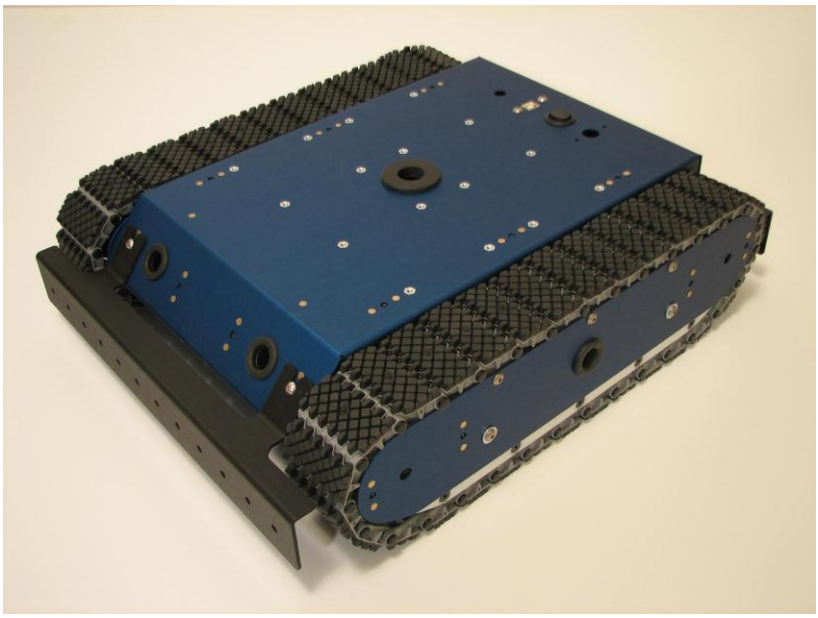

Figure 1: The PUMA robot

Course Integration

First Year Engineering Principles – We have, in the past two years, altered the second semester of the freshman engineering experience to include Chirps as well as C Programming. Students still begin programming with MATLAB, and after 4-5 weeks working with MATLAB, they begin a series of six labs that introduce them to the Chirps system. They begin with basic electronics, assembling and soldering a Raven controller that they get to keep. Students use the Raven to perform pulse-width modulated control of a DC motor and input sensor data. They move into labs that utilize the Owl for sensors and the Woodpecker for DC motor control to understand the Chirps API. Finally, students learn to control the PUMA robot and the course culminates in a robot maze competition. Students have transitioned easily from Matlab to C programming.

Servant Engineering – As part of the engineering curriculum, GFU engineering students participate in a four-semester course sequence where they participate in teams developing service projects for a variety of organizations. Students begin this course in the spring of their sophomore year and continue through the fall of their senior year. One of the challenges with this type of course is the lack of design experience with students at these lower academic levels. The students who were first introduced to the Chirps system in their freshmen course are now in their junior year. As part of their service learning courses, students have utilized Chirps modules to build a variety of devices that they previously would not have been equipped to build. These include a randomized motor controller for a physical therapy device, nursing call system for medically fragile children, and remote control repeaters for a community display. Students have also developed research projects for automated control in a wind tunnel and accelerometer-based angle measuring devices.

Microprocessors – The Chirps system has become the backbone of the junior-year microprocessor course. Each student learns basic microprocessors on a trainer board, and then begins learning how to implement a back-end Chirps system. As the final project for the course,

students are required to develop a unique Chirps system, from conception through schematic design, layout, fabrication, parts ordering and assembly, debug, microprogramming and testing. These students have developed a number of unique modules. Modules include the:

1. Cockatoo – a wired Ethernet interface for the Chirps system.
2. Cockatiel – a wireless Ethernet interface.
3. Pigeon – Location system including GPS, Compass, and accelerometer.
4. Peacock – An LED driver for a matrixed LED display.
5. Budgie – A common interface board for hex displays, LCD display, and button/switch input.
6. Swiftlet – Interface for a number of ultrasonic range finders.
7. Nightingale – an MP3 interface board.
8. Flamingo - The Flamingo board is a VGA controller board that can provide the chirp system with a useful graphical output to either a VGA monitor or an OLED display.
9. Ibis – a weather station controller (humidity, temperature, light, pressure, and a datalogger).

Student boards are currently being built to supplement existing servant engineering, research,and other departmental needs, further extending the Chirps system for future use.

Senior Capstone – Current seniors have made use of various Chirps boards to rapidly prototype systems.

Conclusions

The Chirps system provides GFU engineering students with the ability to design and prototype significant electromechanical systems earlier in the curriculum than they previously could. Course evaluations and student comments have testified to the increased effectiveness of students using this system.

In order to make the Chirps system more effective, we are looking at ways to productize certain boards in the system. We have just completed a new version of the Budgie using surface mount parts rather than through-hole parts. This will allow us to produce a larger number of boards at a lower cost and have them professionally assembled. These boards will be used as the microprocessor trainer boards for the 2011 spring microprocessor course.

We invite the community to participate with us in the development of the Chirps systems. Current board designs and software are available at http://engr.georgefox.edu/Chirps.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0720526. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] G. Spivey, B. Harder, , "Starting a Multidisciplinary Senior Capstone Design Course", 2007 Capstone Design Conference, June 2007, Boulder, CO.

[2] D. Heer , R. Traylor, T. Thompson., and T. Fiez,, "Enhancing the Freshman and Sophomore ECE Student Experience using a Platform for Learning," in *IEEE Transactions on Educations,* vol. 46, no. 4, November, 2003.

[3] T., Thompson, D. Heer, S. Brown, R. Traylor, T. Fiez, "Educational Design, Evaluation, & Development of Platforms of Learning," in *34th ASEE/IEEE Frontiers in Education Conference*, October, 2004.

[4] D. Heer , R. Traylor., and T. Fiez "TEKBOTS: Creating Excitement for Engineering Through Community, Innovation, and Troubleshooting,"in *32nd ASEE/IEEE Frontiers in Education Conference*, November, 2002.

[5] D. Heer, R. Traylor, T. Th'ompson, and T. Fiez, "Integrating Computer Engineering Education with a Platform for Learning," in *33rd ASEE/IEEE Frontiers in Education Conference,* November, 2003.

[6] J. Mossbrucker, O. Petersen, S. Reyer, S. Williams,G. Wrate, , "Addressing the Future: Development of an Electrical Engineering Curriculum," *Proc. American Society for Engineering Education Conference,* 2005.

[7] J. Lenoir, "The Wobbler Steam Engine: A Connection Between the Past, Present, and Future of Mechanical Engineering," *Proc. American Society for Engineering Education Conference,* 2004.

[8] J. Natzke and N. Ninteman, "The George Fox University Freshman Experience: A Projects Based Integrative Approach to Engineering Design," *Proc. American Society for Engineering Education Conference,* 2005.

[9] G. Spivey, J. Bader, N. Ninteman, "Work In Progress: Building MatBot – A Platform for Freshman Robotics with Use Throughout the Engineering Curriculum " in *Proc. Frontiers in Education Conf.,* 2007, pp. S1J-12 - S1J-13 .

[10] D. Jolliffe, "Arduino Fever" *Make: technology on your time,Vol. 7,* pp. 52-53.