AC 2008-1177: PUTTING THE ENGINE BACK IN THE ENGINEER

Fred Cady, (Retired) Montana State University

Fredrick Cady is a Professor Emeritus in the Electrical and Computer Engineering Department, Montana State University. He has been involved with ABET accreditation for the Electrical Engineering and Computer Engineering programs at Montana State University for 20 years. He is interested in improving the quality of engineering education and has authored four microcomputer textbooks. He has a Ph. D. in electrical engineering from the University of Canterbury, NZ and is a senior member of IEEE.

John McLellan, Freescale Semiconductor

John McLellan is a applications engineer for the University Programs at Freescale Semiconductor. He currently works with universities, authors, and industry partners around the globe to drive, create, and implement student learning tools and curricula which support Freescale products in the classroom. John has a B.S. in Electrical Engineering Technology from Texas A&M University.

Putting the Engine back in Engineering

Abstract

Electrical and Computer Engineering programs across the nation are seeing a decrease in engineering student enrollment and retention. Electronic systems and components these days are far too small and complex to allow an inquisitive student to explore and satisfy their curiosity about how these gadgets work. These students often take to exploring mechanical systems instead and are thus led away from Electrical and Computer Engineering. Similarly, with advances in computer simulations of engineering circuits and models that produce realistic results, engineering programs have transitioned away from physical hardware and hands-on experimentation. This trend away from having students being able to "tinker" with real hardware is detrimental to their development into well rounded engineers. In addition, as globalization continues, engineers must broaden their team-work and technical skills.

This paper describes a hardware, software and courseware learning ecosystem that has been created to capture student attention and develop a broader skill set. Laboratory and in-class exercises use POGIL (Process Oriented Guided Inquiry Learning) – based laboratory modules to engage students in learning through exploration, critical thinking, and team and cooperative participation exercises. Laboratory and in-class exercises are designed to teach the student how to explore a new technology to be able to learn more about it. In fact, learning how to learn is a key outcome. Laboratory hardware is designed to provide easy connection to real-world devices and allow students to extend their explorations from classroom theory to the practical application of technology they are learning.

Introduction

Over the past few years, educators have seen a technical boom in the semiconductor industry with more computational power packaged into the smallest of packages. It is becoming less feasible for hands-on, practical-application oriented courses to have students fiddling with hardware at the processor level. In addition to this fundamental size problem, new generations of computational hardware are appearing far faster than we can update laboratory facilities. As we scramble to keep up with technology changes, we find ourselves still behind. Structuring hardware, software, and courseware into reusable components allows the core knowledge to grow alongside technology advances. This type of courseware is not only re-usable, but fosters skills needed for upcoming generations of engineers in a cooperative learning environment.

First, we will discuss the adoption of a teaching methodology which spurs creative, scientific, and collaborative thinking. Initial care has been taken to re-think the way we write course materials so that they are more easily adapted to the changes in technology. With this approach we are able to encapsulate the fundamental information and quickly and easily apply to the latest hardware.

Next, we will show how the Freescale Student Learning Kits embrace an adaptive, modular and reusable teaching platform. Modular hardware boards (Application Modules), a project board, and CodeWarrior™ software tools provide a flexible, common platform that can be applied to a

spectrum of courses. We will show how this teaching platform offers the students more hands-on experience using hardware and how the teaching faculty can encapsulate knowledge into functional blocks and greatly extend the life expectancy of teaching material investments.

Cooperative Learning Approach

The idea that learning is enhanced by learners cooperating with one another goes back over a century with considerable research having been done over the last 20 years[1]. A number of university departments/learning centers have been established to research and promote new techniques to enhance the learning experience of the student[2,3,4,5,6,7,8,9,10,11,12]. Cooperative learning, collaborative learning, active learning, ConcepTests, Process Oriented Guided Inquiry Learning (POGIL)[11] and others subscribe to the idea that "two heads are better than one". While they all have similar strategies and techniques for getting students together in groups, we were struck by the ideas presented by the POGIL approach (http://www.pogil.org).

POGIL

The POGIL process follows the general ideas of many of the other cooperative learning strategies. It promotes breaking the students into learning teams with students assuming roles such as team leader or manager, presenter or spokesperson, recorder, reflector or strategy analyst, and technician. The teams then undertake an activity, perhaps after some initial classroom lecturing, where they are given an activity assignment that includes the following sections:

- A clear, inspiring and communicative title.
- A "why" section to put the activity into context for the student.
- A list of prerequisites.
- Two or three clear and concise statements of learning objectives.
- The information, or model, that the students are to explore to be able to meet the objectives.
- Key exploration and concept invention/formation questions.
- Skill exercises.
- Problems or applications requiring higher-level thinking skills.
- A closure including self-assessment and reflection on learning.

The POGIL technique has been successfully implemented in the teaching of general chemistry in several universities[13]. POGIL has received NSF Course, Curriculum, and Laboratory Improvement (CCLI) funding to continue the growth of POGIL implementation throughout the country by presenting cost-free workshops, funding the development of new teaching materials, and evaluation of student learning[14].

While a true POGIL implementation replaces the traditional lecture classroom with POGIL activities, we were intrigued by the exploration part of the POGIL process and how it might improve the student's laboratory experience. In thinking about how we, the educator, prepare ourselves to be able to teach our students a new microcontroller, for example, we realized that this exploration process, encapsulated so well by POGIL, is exactly what we have been doing for years whenever we have to learn a new topic. When we have a topic that we have to learn, say

interrupts, we open the microcontroller's reference manual to the interrupt chapter and read the information. We form a model of how we expect the microcontroller to behave and then invent small programs that allow us to test our understanding. We expand these simple programs to more complex applications after assessing and evaluating our understanding of the topic. In our old professor-standing-up-at-the-board-lecturing method, we then go and "teach" this to the students by telling them what we have learned. In most of our rapidly changing technology fields this "taught" knowledge is transitory and shortly the students will have to learn a new technology. Wouldn't it be much better for us to "teach" the students the material we wish them to learn while teaching them how to learn it the way that we just learned it? This is the beauty of the POGIL approach. It not only encourages and practices effective cooperative learning, but it also allows the students to learn a body of knowledge (how this microcontroller works) plus how to learn about the next one that comes along (which is the real knowledge gain).

The "G" and the “I” in POGIL are for “Guided Inquiry”. When students are confronted with an entirely new subject they don’t even know how to ask good questions. However, having been there and done that, we do. For example, we know that we have to find out from the hardware engineer the program memory addresses so we can locate our program properly to run on this hardware. You, as the expert, have been through the learning process, at least once. Your responsibility is to be able to guide the students along an effective path through all the information that is available.

Our POGIL Modules

We have developed a collection of POGIL-like modules for two different microcontroller families[15]. After starting with the popular 16-bit Freescale HCS12 family, it was a fairly straightforward exercise to convert these modules to both 8-bit and 32-bit equivalents using the Freescale Flexis family. The Flexis family ranges from the MC9S08, 8-bit, low-power microcontroller to the MCF51 ColdFire 32-bit processor. Because of the very nature of this microcontroller architecture, an instructional module written for one family member is, for the most part, directly useable for another. Due to some of the advanced capabilities of the 32-bit ColdFire architecture there has had to be some extra effort spent in pointing out these differences.

These POGIL-like modules are most suited for use in a laboratory course but elements of them can be used in the classroom. In fact, starting in the classroom and then moving to the laboratory would be most effective. These modules are not like the usual cook book laboratory exercises because they challenge the students to look for the information needed to learn about the topic and then to prove in some way that they understand it. Student collaboration is expected and many of the modules require a student laboratory group to visit with other groups to compare solutions and understanding.

We have written each module to include both a student and an instructor edition. The student edition is freely available on the Freescale University Programs website[15]. The instructor edition materials include solutions as well as sidebar lesson and lecture commentary that need granted access to view. Validated instructors are granted access into an exclusive portal that includes module instructor editions, completed software examples, and a POGIL template to use to create

their own modules. See Appendix I for a list of POGIL module topics developed to date and Appendix II for a sample POGIL module.

Why the need for Modular Teaching Platforms

For years, industry has been using model-based design to build upon existing intellectual property (IP), thereby creating a collection of interchangeable IP blocks which can be easily adapted to new products. As manufacturing processes improve, the implementation of these blocks gets smaller and smaller. This miniaturization and densely packed integration, while good for consumerism, poses an educational roadblock to inquisitive young minds. It is now much more difficult for the students to see the component parts of these complex systems. In order to gain lost ground our first objective is to un-embed technology by creating a system which brings the technology back to a level where there are hands-on opportunities to tinker, reconfigure, and re-engineer. Second, by encapsulating course exercises, circuits, and demonstrations with an upgradeable platform we can retain key blocks of knowledge which ultimately require less time to modify courses when updating laboratories.

Student Learning Kits

Freescale Student Learning Kits (SLK's) are complete hardware and software development toolkits designed specifically for education. The kits highlight Freescale's broad portfolio including microcontrollers, digital signal processors, and wireless connectivity as seen in Table 1. The selection criterion has taken into account such factors as commercial popularity, reference material availability, third-party resources and available feature set.

Table 1 Microcontroller Application Modules

Processor	Architecture	RAM (kB)	Flash ROM (kB)	Bus Clock (MHz)
HCS08QG8	HCS08	0.512	8	10
MC9S12C32	HCS12	2	32	25
MC9S12DT256	HCS12	12	256	25
HCS12XDT512	HCX12	20	512	40
DSP56F801	DSP	4	24	80
MCF5211	ColdFire V2	16	128	66
MCF5223	ColdFire V2 Ethernet	32	256	60
MC13192U	ZigBee RF			
MC9S08QE128	Flexis HCS08	8	128	25
MCF51QE128	Flexis ColdFire V1	8	128	25

Each student learning kit contains the Application Module[16] and CodeWarrior™ software development tools. The Application Modules typically have a short-listed feature set packaged in a small form factor, possess a standard expansion interface, and are fully operable. CodeWarrior is a leading software integrated development environment (IDE) for editing, compiling, and debugging embedded software on Freescale microprocessors.

Unlike traditional all-inclusive development boards, the Application Modules encourage external circuit, peripheral, or system integration. By leveraging the standard expansion interface, the educator can create re-usable labs, circuits, and materials that do not need to be redesigned over

time. To emphasize this point, we can design labs for motor control, audio, keyboard, or power switching circuits which can be reused regardless of Application Module selection.

To further complement the Application Modules, Freescale has developed the Project Board. The Project Board includes all the user features you are accustomed to seeing such as switches, LEDs, displays, multiple power options, and on-board Background Debug Mode (BDM). More prominent, are the large prototyping area and interface connection to National Instrument ELVIS and other third-party instrumentation tools. These provide the educator and equipment administrator a static platform upon which to create curricula, exercises, and equipment setups.

By embracing the benefits of this static platform, one can equip a lab that addresses multiple fundamental courses such as digital logic, filter design, and circuit analysis in addition to microcomputer design, embedded system interfacing, and control systems as shown in Figure 1. Moreover, by creating component "building blocks" one can quickly adapt these components to work with new technologies. This time savings environment allows us to focus on learning multiple new technologies. Also, when a microcontroller module can be used in more than one course, as shown in Figure 1, the overall equipment costs can be less.

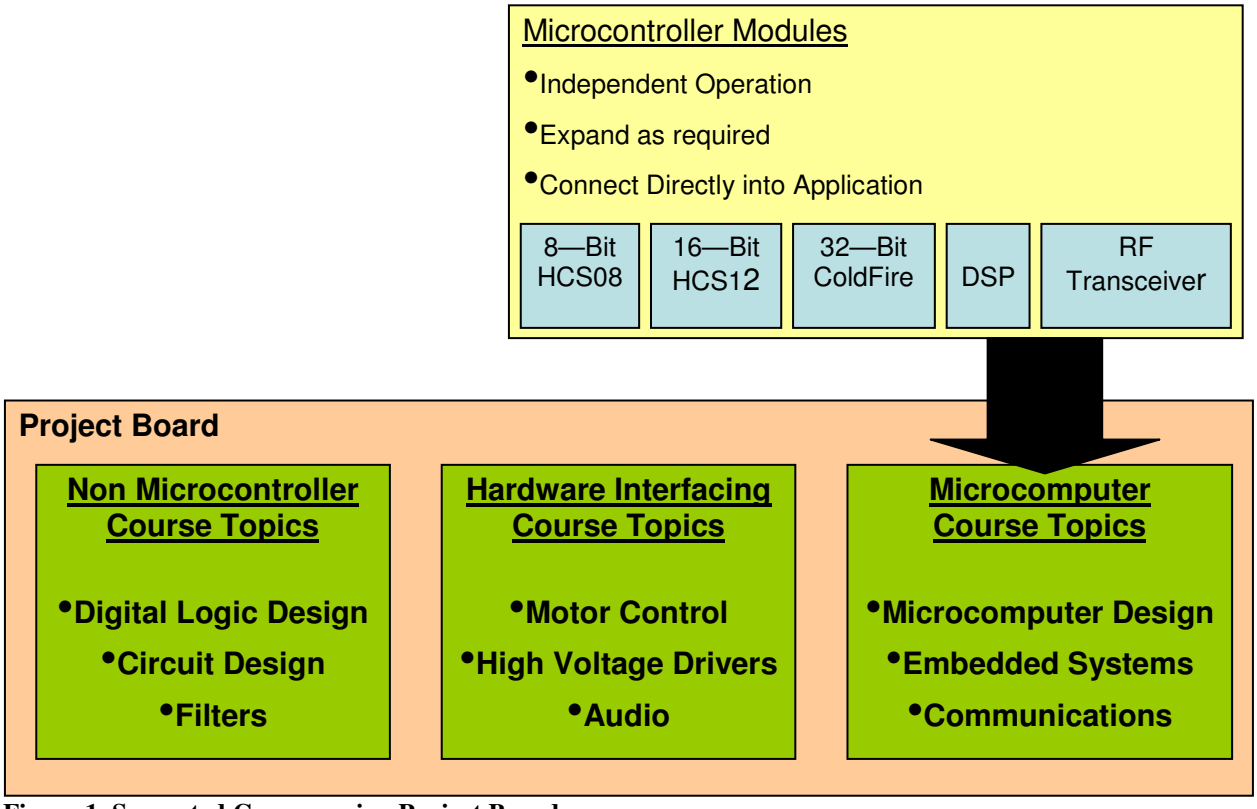

Figure 1. Suggested Courses using Project Board

Student Learning Kit Use at Montana State University

The student learning kits are used in a sequence of three microcontroller embedded system classes in the Electrical and Computer Engineering Department at Montana State University. HCS12, HCS08 and Zigbee ready RF-Transceiver modules are used with the project board.

Although Computer Engineering students are not required to purchase the student learning kits, many do because all three embedded courses are required for their degree. Many of the Electrical Engineering degree students, who are required to take only the first embedded course, also enroll in the subsequent classes and they purchase their own kits too. Our second embedded class requires students to learn about the ZigBee ready RF-Transceiver modules and to implement a RF-based mesh network environment for sharing data. This scheme allows the students to work on their laboratory projects outside of laboratory class time. It also spreads the cost of the student learning kit over three courses. To support students who do not purchase a kit, our introductory course laboratory is equipped with HCS12 modules and project boards. In addition, the Department subsidizes the initial cost of the kits purchased by the students. The student learning kits are also used extensively in our senior capstone project course. Depending on the instructor, some of the introductory laboratory assignments have adopted the POGIL approach assigned above.

It is early days in our implementation of the POGIL methods in our laboratories. Although no formal assessment has been done at our institution, as it has at other universities[17], we are finding that after an initial resistance to the change in paradigm the students are generally preferring this method over the more traditional laboratory exercise course.

Conclusion

By applying a three-fold approach to how we design course material, hardware and software for curricula we can create a library of re-usable, modular components for teaching. Embracing this methodology in hardware and software tools can be realized through Freescale Semiconductor Student Learning Kits. By skillfully creating courses in this manner, we can focus on producing quality, tried and true, building blocks which engage students and un-embed complex systems. Students learn how to learn, and get a low-cost, multi-purpose tool that they can create there own building blocks to use throughout their life-long learning. Finally, the effective use of the described approach greatly increases the ability, cost and speed of incorporating new technologies or additional blocks.

Bibliography

1. Roger T. and David W. Johnson, *An Overview of Cooperative Learning*, originally published in J. Thousand, A. Villa and A. Nevin (Eds), Creativity and Collaborative Learning; Brookes Press, Baltimore, 1994., available on-line at http://www.co-operation.org/pages/overviewpaper.html.
2. Kennesaw State University Educational Technology Training Center, http://edtech.kennesaw.edu/
3. The Cooperative Learning Center at the University of Minnesota, http://www.co-operation.org/.
4. Active/Cooperative Learning: Best Practices in Engineering Education, Arizona State University, http://clte.asu.edu/active/main.htm
5. International Association for the Study of Cooperation in Education – IASCE, http://www.iasce.net/
6. National Institute for Science Education, University of Wisconsin, http://www.wcer.wisc.edu/archive/cl1/
7. Project Galileo, Harvard University, http://galileo.harvard.edu/
8. Center for Teaching, Vanderbuilt University, http://www.vanderbilt.edu/cft/resources/teaching_resources/activities/cooperative.htm
9. The Cooperative Learning Network, Sheridan College Institute of Technology and Advanced Learning, http://www-acad.sheridanc.on.ca/scls/coop/cooplrn.htm
10. New Horizons for Learning, http://www.newhorizons.org/
11. Process Oriented Guided Inquiry Learning, http://www.pogil.org

12. Many others!
13. Franklin and Marshall College, Stony Brook University, Washington College, Portland Community College, Berry College, Boise State University, and others.
14. http://www.pogil.org/info/grants.php
15. http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=06258A2580257B
16. Application Module refers to a hardware development tool
17. http://www.pogil.org/effectiveness/

Appendix I

Module Name	Microcontroller	Title
LABS12CINTRO01	Any	The Microcontroller - General Principles
LABS12CINTRO02	Any	General Principles of Software Development
LABS12CINTRO03	HCS12	Introduction to CodeWarrior – Simulating the Microcontroller in Assembly Language
LABS12CINTRO04	HCS12	Introduction to CodeWarrior - Running Assembly Programs on the Microcontroller
LABS12CINTRO05	HCS12	The Assembler
LABS12CINTRO06	HCS12	Exploring Embedded C Programming
LABS12CINTRO07	HCS12	Introduction to CodeWarrior™ – Simulating the Microcontroller in C
LABS12CINTRO08	HCS12	Introduction to Your Microcontroller Hardware
LABS12CINTRO09	HCS12	The Microcontroller Instruction Set I
LABS12CINTRO10	HCS12	The Microcontroller Instruction Set II
LABS12CINTRO11	HCS12	The Timer – Introduction to Timer Overflows With C
LABS12CINTRO12	HCS12	Digital Input and Output
LABS12CINTRO13	HCS12	Digital Input and Output With C
LABS12CINTRO14.	HCS12	I/O Software Synchronization
LABS12CINTRO15.	HCS12	Introduction to Interrupts Using C
LABS12CINTRO16	HCS12	Introduction to Interrupts
LABS12CINTRO17	HCS12	The Timer – Introduction to Timer Overflows
LABS12CINTRO18	HCS12	The Timer – Timer Overflow Interrupts
LABS12CINTRO19	HCS12	The Timer – Output Compare
LABS12CINTRO20	HCS12	The Timer – Input Capture
LABS12CINTRO21	HCS12	The Timer – Pulse Accumulator
LABS12CINTRO22	HCS12	Analog Input
LABS12CINTRO23	HCS12	Sampling and Resolution for Analog Input
LABS12CINTRO23	HCS12	Sampling and Resolution for Analog Input
LABS12CINTRO24	HCS12	HCS12 A/D Digital I/O
LABS12CINTRO25	HCS12	Register Listing HCS12C Family
LABS12CINTRO26	HCS12	Analog Input in C
LABS12CINTRO27	HCS12	The bouncing Switch in Assembly
LABS12CINTRO28	HCS12	The Bouncing Switch in C
LABS12CINTRO29	HCS12	Serial I/O – SCI
LABS12CINTRO30	HCS12	Serial I/O Interfaces – RS-232-C
LABS12CINTRO31	HCS12	Serial I/O – The Serial Peripheral Interface
LABS12CINTRO32	HCS12	Sources of Multiple Interrupts
LABS12CINTRO33	HCS12	Computer Operating Properly – The COP
QEACADLABS1	Flexis Family	DEMOQE128 – Getting Acquainted
QEACADLABS4	Flexis Family	Introduction to the Flexis Microcontroller Hardware
QEACADLABS2	MC9S08	Introduction to CodeWarrior™ – Simulating the MC9S08 Microcontroller in C
QEACADLABS3	MC9S08	Introduction to CodeWarrior™ – Running the MC9S08 Microcontroller in C
QEACADLAB19	MCF51	Introduction to CodeWarrior™ – Running the ColdFire Microcontroller in C
QEACADLABS5	MC9S08	Exploring Embedded C Programming for the MC9S08
QEACADLABS20	MCF51	Exploring Embedded C Programming for the MCF51
QEACADLABS6	Flexis Family	Digital Input and Output With C
QEACADLABS11	MC9S08	Introduction to MC9S08 Resets and Interrupts

QEACADLABS12	MCF51	Introduction to MCF51 Resets and Interrupts
QEACADLABS13	Flexis Family	The Timer – Introduction to Timer Overflows
QEACADLABS14	Flexis Family	The Timer – Introduction to Timer Overflow Interrupts
QEACADLABS8	Flexis Family	The Timer – Output Compare
QEACADLABS7	Flexis Family	The Timer – Input Capture
QEACADLABS10	Flexis Family	Multiple Interrupt Sources
QEACADLABS9	Flexis Family	The Bouncing Switch
QEACADLABS15	Flexis Family	Serial I/O – SCI
QEACADLABS18	Flexis Family	Serial I/O – The Serial Peripheral Interface
QEACADLABS16	Flexis Family	Analog-to-Digital Conversion
QEACADLABS17	Flexis Family	Computer Operating Properly

Appendix II Sample POGIL Module

Instructor's Version. A student version will not have the rightmost column or the Instructor's Notes.

Digital Input and Output

Overview A microcontroller must be connected to external devices to be able to do any useful work. A typical embedded application would have the microcontroller receiving information (inputting) from an *input device*, modifying or making decisions based on the information and the task at hand, and outputting some control action or information to an *output device*.	**Lesson Planning:** This module starts the student using the I/O capabilities of the microcontroller. It does not use interrupts.
Learning Objectives This module will help you learn about the parallel input and output capabilities of your microcontroller.	
Success Criteria When you have completed this module you will be able to demonstrate that you can retrieve information from an input device and output to an output device.	
Prerequisites You must be able to write assembly or C programs for your microcontroller and be able to test and run programs on a student learning kit in the laboratory.	
More Resources and Further Information Cady, Fredrick M., *Software and Hardware Engineering: Assembly and C Programming for the Freescale HCS12 Microcontroller*, 2nd edition. (New York: Oxford University Press, Inc., 2008), Chapter 11 HCS12 Parallel I/O *MC9S12C Family, MC9S12GC Family Reference Manual HCS12 Microcontrollers,* Freescale Semiconductor, Austin, Texas, December 2006 (mc9s12c128v1.pdf)	
I/O Ports All microcontrollers have ports that may be used to input or output digital (binary) information. In some cases a port may be restricted to be either an input or output port exclusively. For most ports in modern microcontrollers you may choose the direction of information flow and program the port to operate in either mode. In some cases, this choice can be made for individual bits within the port, giving us the capability of having both input and output bits on the same port. When the microcontroller is reset, all bidirectional ports are initialized to be in the input mode. After reset your program must choose the operating mode for the ports.	
Explore 1. 1. Make a table listing the ports, the port address, and whether or not the port can be bidirectional. (Make a table with six columns to add other information in later explorations.)	**Answers:** See Instructor's Notes below.
Stimulate 1.	**Answer:**

1. Why do microcontroller bidirectional ports operate as input ports when the microcontroller is reset, even though they may be connected to output hardware?	1) This is the safe hardware mode. If the port is initially an output and it is connected to input hardware, then two outputs are connected together. Depending on the relative source impedances the devices may be damaged.
Bidirectional Control Ports with the capability of being either input or output use a control register called the *data direction control register*. Bits in this register are programmed to control the direction of each bit in the data port.	
Explore 2. 1. Add the name and the address for the data direction register of each bidirectional port in the table started in Explore 1. 2. How must the data direction control bit in the data direction register be initialized so that the bit is an output? An input?	**Answers:** 2) 1 = output, 0 = input.
Explore 3. Your laboratory student learning kit hardware contains input devices, such as pushbutton, or dual in-line package (DIP) switches, and output devices such as LEDs. Inspect your hardware documentation and add to the table started in Explore 1 information showing what hardware is connected to which microcontroller port. You may need to specify the function of individual bits within the port.	**Teach:** This table will give the student a resource to be used in future labs.
Stimulate 2. Demonstrate that you understand the principles of digital I/O by designing and writing an I/O program. Do this in the following sequence of steps: 1. Write a software requirements specification that states what the program is to do. 2. Write a software design that shows exactly what the program must do to accomplish the written specification. The design should be written as comments to be included as part of your program. Follow top down design principles and make sure you have a fully designed program before inserting instructions beneath each design statement. *Example:* *; Initialize port_x, bits xyz to be input and port_y, bits pqr to be output* *; Input switch positions from port_x* *; Output data to port_y* *; etc.* 3. Insert microcontroller instructions that will accomplish the design step beneath each design comment. As you proceed you may find that other design comments and code must be included to make your program work. That is OK. Add anything to the design you need and then add the instructions.	**Teach:** You may wish to guide the students to help them with a requirements specification. *Example: This program is to input the current configuration of the switches and to display that configuration on the LEDs.*

4. Demonstrate that your program works as designed and meets the requirements.	
Communication – Inter-Group Compare your solution to Stimulate 2 with another laboratory group's solution. Have they chosen a different way to demonstrate their understanding of parallel I/O? Can you understand their program by reading the design comments and the inserted code?	
Reflection on Learning Did your program work perfectly the first time you tried to run it on the student learning kit? If not, did you have to add design features you didn't think of, or did problems arise because you did not use the correct instructions to implement the design? Could you have foreseen the problems and included the fix in the design phase? If so, what would you have had to know to be able to do this?	
Communication – Reporting Prepare a memo report for the laboratory instructor. Include comments gained during your Inter-Group Communication and Reflection on Learning. Include listings of your and the other group's programs. How does each solution compare in terms of meeting the requirements specification, design comments, and use of instructions to implement the design. Did you learn anything from seeing the other group's solution?	

Instructor's Notes

Explore 1 Answers

(Answer deleted.)

Stimulate 2 Sample Program

(Sample program deleted.)