Using Information Gap Learning Techniques in Embedded Systems Design Education

Dr. J.W. Bruce, Mississippi State University

J.W. Bruce is an Associate Professor in the Department of Electrical & Computer Engineering at Mississippi State University.

Mr. Ryan A. Taylor, Mississippi State University

Mr. Ryan Taylor is currently a doctoral candidate in the Department of Electrical and Computer Engineering at Mississippi State University. He received his BSEE and MSEE from the University of Alabama, where his thesis centered on microcontroller education tools. His doctoral research focuses on asynchronous circuit synthesis. In the past he has served as a graduate research assistant at Mississippi State University as well as the instructor of record of multiple courses at both UA and MSU.

Using Information Gap Learning Techniques in Embedded Systems Design Education

Introduction

Commercial market trends tend to trickle into engineering program curricula. In the computing systems marketplace, customers are demanding ever more complex features as computing systems become more capable and affordable. Today, engineering educators are feeling the pressure to provide more realistic, comprehensive, and complex lab experiences to the students in order to remain relevant and keep students' attention. These demands are especially difficult in the university environment where students may lack several basic skills and the professor and student work under an intense 15-week time-to-market.

In order to take students from neophyte to accomplished designer of embedded systems, operating systems specialized for embedded systems may be used to offload many housekeeping and flow control tasks. Many wonderful embedded systems operating systems exist; however, some commercial offerings are cost-prohibitive while others are simply too feature-rich to deploy in the classroom during a typical semester-long course. To this end, the authors have developed a capable, but relatively simple cooperative multi-tasking operating system targeted at resource-limited systems.

A benefit to using an operating system in an embedded systems course to develop embedded systems applications is to teach students to work within the confines of a system in which they may not have full control. As embedded systems become increasingly complex, subsystems are increasingly hidden and black box design skills are required. Use of our OS emulates these design constraints, but unlike commercial OS-es, professors and lab assistants have full knowledge of the OS behavior and can provide detailed support and guidance impossible if the OS were acquired from commercial companies or the open-source community. Once students have a firm understanding of the high-level use of the OS, hardware and software interfacing issues can be fully explored in the alb by requiring the OS be modified and extended by the students. In this manner, students gain a fuller understanding of the limits and the technical reasons for the limits of the OS.

During the implementation of this system into the course, it was noticed that students seemed to spend a large majority of their time working on issues that had been covered in earlier coursework and did not relate directly to the material being covered in the embedded systems lecture. A gap learning technique was introduced to combat this issue. Its goals were to increase the amount of time that a student spent focusing on the concepts being addressed in the embedded systems course and to decrease the amount of time that a student spent working on

issues unrelated to embedded systems. These techniques saw some success and the findings of the investigation are presented here.

The Background section introduces the embedded systems course in use at Mississippi State University, as well as the corresponding lab and its content and tools. The Gap Learning Techniques section present the ideology behind this investigation and the Assessment section discusses the results of their implementation. The Conclusions section draws these results to their implications.

Background

Over the years, the authors have reported on the evolution of a senior-level embedded systems design course at Mississippi State University. The Bagley College of Engineering at Mississippi State University has required engineering students to purchase laptops since 1999, and many courses make substantial use of these valuable student-provided resources. The mission of the embedded systems course described in this paper is to be a miniature cumulative design experience focused on microprocessors and embedded systems. This course is to be a directed, and reasonably tightly controlled, design experience to prepare students for the much more unstructured design experience in senior design. A secondary mission in the course is to introduce the students to practices commonly found in the commercial and industrial embedded systems design environment. Since embedded system designs are so naturally adapted to the object-oriented design paradigm, the course has always tended toward a high-level, reusable code approach.

The initial offering of the course used a specialized (read: expensive) development board with a high-performance 8051 microcontroller running a custom Java virtual machine. The second offering of the course was altered to deal with the high costs and fragility of the development board. This offering adopted a student-developed design for microcontroller interfaces to the hardware and sensors. Students developed the system on a breadboard and eventually created their own professionally-manufactured printed circuit boards. Parts were obtained from parts kits purchased by students in the prerequisite microcontrollers course (Reese 2005). The low-cost 8-bit microcontroller ran C code and provided the communications link to the PC running Java as in the earlier offering. The client-server architecture from the first course remained in place. The third and fourth offerings adopted a cooperative multi-tasking operating system (written by one of the authors) on the 8-bit microcontroller to facilitate faster firmware development cycles. The third and fourth offerings were identical with the exception that the third offering used a traditional RS-232 communications interface to the PC while the fourth version used a USB-based model of the 8-bit MCU.

The addition of the operating system in the third and subsequent offerings of the course was driven by the desire to complete complex designs within a standard academic semester without unduly overworking the students. The lab project (Bruce, Harden, and Reese 2004; Bruce 2004; Bruce and Goulder 2005) would be difficult to complete in a normal academic semester if students were expected to design, write, and test all hardware and software components of the project themselves. Furthermore, an industrial project of this magnitude would likely rely on existing designs or software libraries. To this end, the authors wrote a simple real-time multitasking operating system named the Embedded Systems Operating System (ESOS) based on the very clever protothreads library (Dunkels 2017) by Adam Dunkels.

Protothreads provide a nearly zero-overhead (and stack-free) implementation of lightweight threads (similar to co-routines). Protothreads differ from normal threads in that they can not be preempted and are stack-less. Protothreads provide event-driven flow control in a very readable fashion and a machanism for "blocked" protothreads to give up processor focus so other protothreads can run. Protothreads facilitate very complex flow control without resorting to full state machine software structures. The protothread implementation relies on a Duff's device ("Duff's Device - Wikipedia" 2017), a loop-unwinding technique as applied to the C language case-switch construct. Duff's device seems a bit counterintuitive at first, and may seem like a C language hiccup. However, the Duff's device behavior is mandated by the C language specification. Therefore, Duff's device, and protothreads in turn, are readily implemented in nearly every quality C language compiler. In short, ESOS is exceedingly portable to other compilers and processors. This was evidenced by the porting of ESOS from the 8-bit processor to the 16-bit processor in the fifth offering of the course. Furthermore, the OS has been ported to the PC (both Windows and Linux) to facilitate development of the OS and demonstration in the lecture setting.

The fifth offering of the course was identical to the third except the microcontroller was changed to a 16-bit model and the Python back-end was mandatory. Transitioning from an 8-bit to a 16-bit MCU for hardware interfacing is potentially a very dramatic change. Surprisingly little changed from the students' perspective because the MCU's cooperative OS was ported to the new processor. The operating system API hid much of the change of the data size beneath and only minor documentation changes were required of the instructor. Some small administrative changes were applied to the laboratory milestones for the sake of modularity and continuity.

The sixth offering of the course was identical to the fifth offering of the course, though some aspects of the milestone requirements were made slightly more complex and the length of assignment of some of the milestones were extended to account for this modification. The ordering of the milestones was not modified.

The seventh offering of the course was identical in all aspects to the sixth offering of the course, other than a swap of the ordering of two of the milestones requirements.

Table 1. Milestone content for final three offerings of the course.

Milestone	Offering 5	Offering 6	Offering 7
1	Introduction	Introduction	Introduction
2	OS and Hardware	OS and Hardware	OS and Hardware
3	UI Service	UI Service	UI Service
4	Sensor Service	Sensor Service	Sensor Service
5	LCD Module	LCD Module	LCD Module
6	Digital Function Synthesizer	Digital Function Synthesizer	Serial Communication
7	Serial Communication	Serial Communication	Digital Function Synthesizer
8	Controller Area Network (CAN)	Controller Area Network (CAN)	Controller Area Network (CAN)
9	CAN System Integration	CAN System Integration	CAN System Integration

As the course has matured over the last three semesters, the material has been kept largely the same to investigate the gap learning techniques discussed in this paper. As an added benefit, the support libraries and hardware being used in the lab have matured, allowing for more robust systems as time has moved forward. During the fifth offering of the course, students saw hardware issues with the implementation of the CAN hardware during milestones eight and nine. By the sixth offering of the course, these hardware problems were partially worked out with workarounds, so students were able to complete assigned tasks through the eighth milestone before reaching some integration problems that were related to more CAN hardware issues. Once again, workarounds were found between course offerings, and the students successfully completed all nine milestones during the seventh offering of the course.

Gap Learning Techniques

A noticeable but troubling issue arose after adopting ESOS in the lab portion of the embedded sysetms course. Students were able to fully use the peripherals and services provided by ESOS. However, they often had trouble describing what exactly the service or peripheral did, how it worked, or what was required to initialize it. In short, ESOS sorted out all the details about the particular service, set it up, and performed all the work. Students just had to make the API call correctly. While this is a desired behavior in a commercial or industrial design setting

where a small time-to-market is crucial, it is somewhat discouraging to us as educators that students do not understand basic concepts of embedded systems operations. In short, our goal in the course is to have students learn to trust and use the black boxes – the approach required by our systems-oriented design paradigm – but still gain a working knowledge of the concepts that are addressed by the black boxes? How can we structure the lab experience such that students are required to gain an understanding of subtle details, but not get bogged down in the minutia?

Enter gap learning. In order to address these issues as well as issues of high student workloads and high stress levels among students dealing with their first major system project, techniques were explored that could be implemented into an embedded systems curriculum to allow students to understand the inner workings of the black boxes while still becoming used to using them at the abstract level.

Instead of asking students to simply use the API of ESOS to implement the desired milestone requirements in the lab experience, our gap learning approach is that the authors develop a design framework for each milestone. Once this framework is designed, it is very partially implemented, leaving the large majority of the system unfinished. The omitted portions of the provided design framework is such that diligent study of the framework makes obvious what steps are missing. There is little latitude for the students to do anything else without having to change the provided framework. This approach requires the student to approach the design first with an inquisitorial attitude, searching to understand the framework that has been set up for them. Once this understanding is complete (or sufficient), the student and his or her teammates are able to embark upon the completion of the design requirements.

It is hoped that this technique achieves multiple benefits. First, the techniques will allow the students to see the framework of a successful design before beginning their own implementation. This helps visualize a successful design as a team before they are thrown into the throes of their senior capstone design project. Second, the techniques remove some of the tedious work that should be covered in prerequisite courses. On a quick timeline such as a 15-week semester, precious time cannot be spent rehashing material that the students should be comfortable with already. This also reduces the overall workload from a course that is naturally "work-heavy." Third, these gap learning techniques allow for students to be able to immediately see the heart of the concept that is being addressed with each lab milestone. Added tertiary benefits include a lower stress level for students and potentially smoother team interaction.

As an example of these techniques, consider the integration of a sensor service into the ESOS-driven microcontroller system being used in this course. For reference, this is the service that largely makes up the fourth milestone as described in Table 1. In the initial writeup that the student teams receive, an API is introduced to allow the user to communicate with sensors on board the system. This API is to be designed by the teams and includes such functions as

`ESOS_TASK_WAIT_ON_AVAILABLE_SENSOR` and `ESOS_TASK_WAIT_SENSOR_QUICK_READ`. Generic tasks are to be created by the students that then, in turn, use hardware-specific tasks to carry out their desired functionality (also created by the students.

Simply giving the students these requirements during the fifth and sixth offerings of the course led to a wide variety of implementations: some elegant and efficient, some not. During the seventh offering of the course, to introduce gap learning techniques, the general framework of the generic task was presented to the student. Figure 1 shows an example of this. From here, students were required to analyze the implementation, verify that it correctly implements the desired API functionality, and use it in their version of the code library. Additionally, the students were required to write the hardware-specific aspects of the code that are referenced in the body of the generic task. In the example of Figure 1, the functions `esos_sensor_initiate_hw()`, `esos_sensor_is_converting_hw()`, and `esos_sensor_getvalue_u16_hw()` are all left unimplemented so that the students can complete them.

```
/**
* Waits until a sensor is read (quick version).
*
* \param pu16_data        pointer to the location of the resulting data
*
* \hideinitializer
*/
ESOS_CHILD_TASK(_WAIT_SENSOR_QUICK_READ, uint16_t* pu16_data)
{
    ESOS_TASK_BEGIN();

    esos_sensor_initiate_hw();
    ESOS_TASK_WAIT_WHILE(esos_sensor_is_converting_hw());
    *pu16_data = esos_sensor_getvalue_u16_hw();

    ESOS_TASK_END();
}
```

Figure 1. Example of generic ESOS task given to students.

Assessment

In order to fully investigate the effects of the gap learning techniques described in the previous section, data describing students' performance and exertion was needed from multiple offerings of the course. Fortunately, significant process and product metrics have been collected in all seven offerings of the course. However, as the course has evolved heavily over time, this

paper concentrates on the most recent three offerings of the course to better establish that topic content and order would not influence the results in an overt way. Table 1 describes the order of content presentation during the most recent three offerings of the course.

During these course offerings, students were asked to self-report data relating to their experience in lab throughout the semester. These software metrics included such measures as amount of time spent working on the project, the amount of code contributed to the team repository, the number of errors found during debugging, and others. This data was not only used as a part of this investigation, but also as a part of the ongoing tuning and evolving of the course itself.

It should be noted here that the data reported is self-reported. As in all human-provided data, there exists an element of variation. Some students' will be more hesitant to provide accurate data and others will simply not provide data altogether. This has been accounted for by the normalizing of the data across all students, teams, and work periods throughout each course offering. This will be explained further as the specific sections of data are presented.

The course laboratory sections have been administered by the same teaching assistant in the fifth, sixth, and seventh course offerings. He has been able to observe and interact with the cohorts of students during these three offerings. His responsibilities are to hold weekly meetings with the students as they have matriculated through the milestone requirement sets and built up the final system using the ESOS environment.

During the fifth offering, the TA observed students struggling with heavy workloads in the lab. These heavy workloads transcended the traditional issues that are seen in many group design projects at this level in the curriculum. There were some imbalances among student teams, as is to be expected. However, the workloads were judged to be heavy as the students were required to not only become familiar with the idea of a real-time operating system, but were also being asked (some for the first time) to implement a complex hardware system on a short timetable. The students during this term dealt with high stress levels as they moved deeper into the material. For some students, the high stress situations led to a stronger work ethic and an increased sense of team responsibility. For others, it deepened the divide between members of their team who seemed more ahead of the curve and members who seemed to be lagging behind or not pulling their weight. This had an obvious impact on the final product that was eventually delivered.

During the sixth offering, as the requirements were slightly increased and accelerated for certain milestones of the design requirements, students were observed to show similar traits of the students from the fifth course offering. Overly high stress levels and stressful team meetings took place throughout the semester, reaching a peak during the final few milestones.

Interestingly, the changes that were made to the design requirements did not seem to affect the issues facing the students one way or another. This could be due to the fact that the changes were not significant enough to make a difference. The lack of any relevant change could also be due to the amount of workload not being the root of the issues. It was during this offering that the decision was made to incorporate the gap learning techniques discussed in this paper.

During the seventh offering of the course where gap learning was used, the stress levels of the student cohort was noticeably lower. There was a reduced amount of problems arising during supervised team meetings along with fewer problems related to team member workload equity. During discussions with individual team members it was judged that students had a greater understanding of the concepts that were being discussed in the course. By incorporating these gap learning techniques, students were able to focus a larger percentage of their invested time into the concepts that are covered in the lecture portion of the course and implemented as a part of the lab milestones.

For the purposes of this investigation, student effort is defined as the number of hours that each student worked on the project associated with this course per week. As some of the milestones lasted for longer than one week and some teams were larger than others, the data has been normalized for one student over the course of one workweek. Figure 2, below, shows the student effort for the last three offerings of the course.

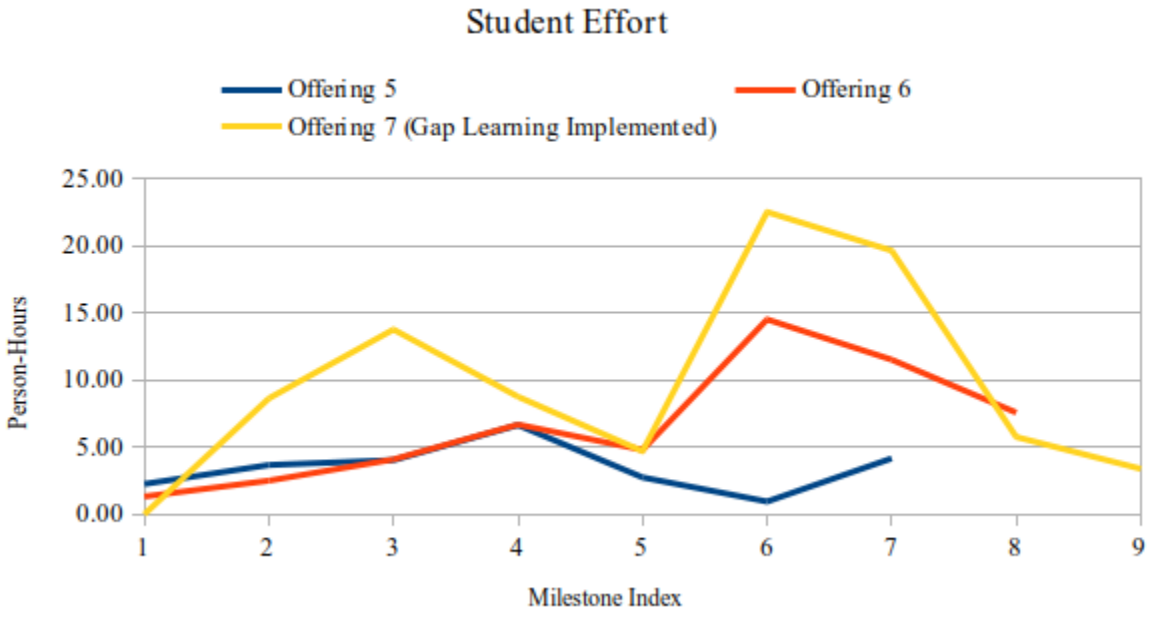

Figure 2. Student effort exerted in the most recent three offerings of the course. Note: milestones six and seven swapped in offerings five and six to facilitate comparison; see Table 1.

As can be seen, during the semester in which gap learning techniques were used, the overall work hours increased relative to the previous two course offerings with similar design lab requirements. The authors had hoped that gap learning would reduce the time requirements on the students in the lab, while maintaining student learning and understanding. Again, the data in Figure 2 is self-reported by the students – different cohorts in different semesters. If the data in Figure 2 is accurate, we hypothesize that more time is required for study of the partially-complete design framework in order to understand the structure of the solution before student development begins. However, it could also be due to the fact that students found difficulty in writing code for an existing solution. Whatever the reason, even when accounting for the natural variation of self-reporting data from multiple humans, the amount of time reported being spent working on these problems seem to be increased when gap learning techniques were used.

For the purposes of this investigation, the measure of how much work output a person produced is quantified by the number of lines of code that a person wrote during the time that was devoted to a particular milestone. Again, the data has been normalized for one student but now working during one single workday. Figure 3, below, shows the lines of code written per student per workday per milestone during the most recent three offerings of the course.

Figure 3. Lines of code written per workday in the most recent three offerings of the course. Note: milestones six and seven swapped in offerings five and six to facilitate comparison; see Table 1.

As can be seen in Figure 3, the average amount of work reported completed increased from offering five to offering six. This is to be expected, as the design requirements became a little more complex between these offerings. It is noteworthy, however, that the average work

reported completed decreased on average from the sixth offering to the seventh offering. This was the desired result of incorporating gap learning techniques, as long as it did not come at the expense of student understanding. Coupled with the data related to student hours expended, this means that students spent much more time designing the system, understanding the system, and thinking about the concepts involved in the system rather than writing code for a microprocessor.

Another metric of interest during this investigation was how the gap learning techniques affected the efficiency of the students. This measure was quantified in the number of software defects that existed in the code written by the student. The reporting of these defects required the students to be specific with the type of defect coded. However, the measure being used here is simply the total number of defects that existed (and were found by one student or another). Figure 4, below, contains the data for this measure.

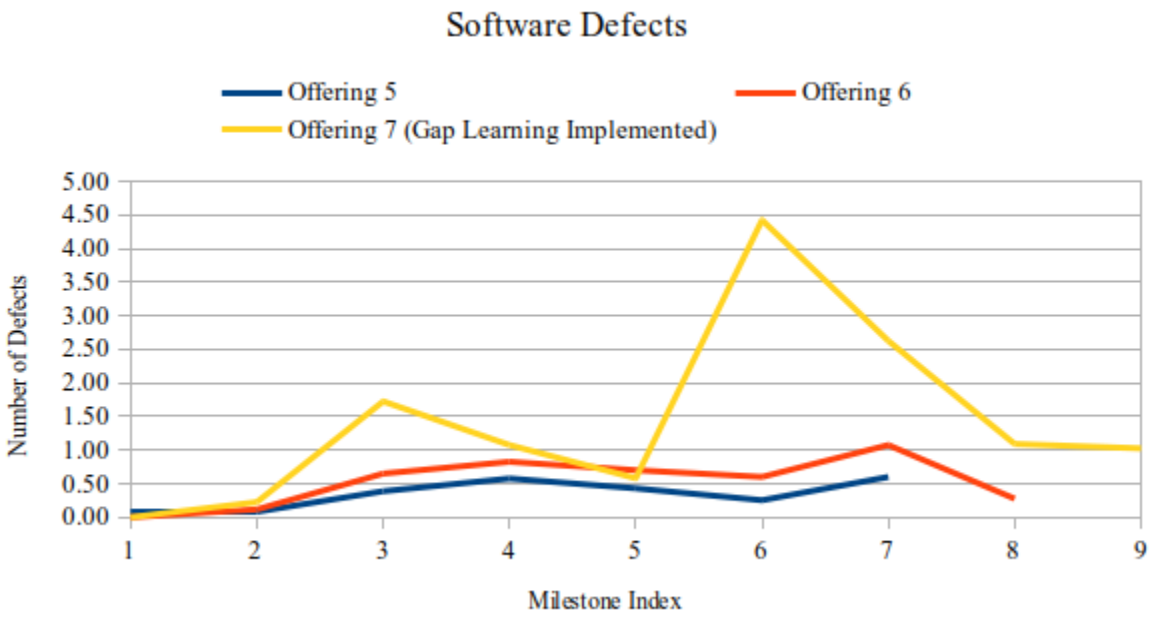

Figure 4. Number of software defects found in a student's code for each measure for the most recent three offerings of the course. Note: milestones six and seven swapped in offerings five and six to facilitate comparison; see Table 1.

On average, the number of software defects reported were much higher in the seventh offering of the course compared to the fifth and sixth offerings. This either indicates that the student was a less efficient coder when gap learning techniques were used in the course or that the student had a more difficult time coping with learning the solution strategies that were used in the gap learning process. Once again, the product measures of defects are student self-reported and may not be comparable to similar data in previous course offerings.

Conclusions

This investigation presented the findings of an investigation into gap learning techniques in the setting of an embedded systems course in an electrical engineering curriculum. The authors provided a partially complete design framework for each significant lab task. Students were expected to study the provided framework, ascertain its intent, and provide the missing components – both hardware and software – to form a working solution. The authors' intent was to reduce student effort in time exerted and student output while maintain, or increasing student understanding. Results are encouraging. While students seem to report spending even more time on the lab portion of the course, the student output measured in lines of code were reduced. The ultimate student designs were at least as successful as previous semesters, but students self-reported more software defects in their product measures. Finally, the ultimate goal of reducing student stress aver lab work loads did appear to be successful, as both the instructor and the lab teaching assistant reported that students seemed more relaxed about requirements.

References

Bruce, J.W. 2004. "Design Inspections and Software Product Metrics in an Embedded Systems Design Course." In *Proceedings of the ASEE Annual Meeting and Exposition*, 9.381.1-9.381.8. Salt Lake City, UT. https://peer.asee.org/13391.

Bruce, J.W., and Jordan Goulder. 2005. "A First Look at an Internet-Enabled Embedded Systems Design Course." In *Proceedings of the ASEE Annual Meeting and Exposition*, 10.38.1-10.38.14. Portland, OR. https://peer.asee.org/14555.

Bruce, J.W., J.C. Harden, and R.B. Reese. 2004. "Cooperative and Progressive Design Experience for Embedded Systems." *IEEE Transactions on Education* 47 (1): 83–92. doi:10.1109/TE.2003.817618.

"Duff's Device - Wikipedia." 2017. https://en.wikipedia.org/wiki/Duff's_device.

Dunkels, Adam. 2017. "Protothreads - Lightweight, Stackless Threads in C." http://dunkels.com/adam/pt/.

Reese, Robert B. 2005. "Embedded System Emphasis in an Introductory Microprocessor Course." In *ASEE 2005 Annual Conference*, 10.525.1-10.525.7. Portland, OR. https://peer.asee.org/15572.