European Journal of Engineering Education
Vol. 37, No. 6, December 2012, 557–574

Taylor & Francis
Taylor & Francis Group

Embedded C programming: a practical course introducing programmable microprocessors

David M. Laverty*, Jonny Milliken, Matthew Milford and Michael Cregan

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, UK

(*Received 26 January 2011; final version received 21 August 2012*)

This paper presents a new laboratory-based module for embedded systems teaching, which addresses the current lack of consideration for the link between hardware development, software implementation, course content and student evaluation in a laboratory environment. The course introduces second year undergraduate students to the interface between hardware and software and the programming of embedded devices; in this case, the PIC (originally peripheral interface controller, later rebranded programmable intelligent computer) microcontroller. A hardware development board designed for use in the laboratories of this module is presented. Through hands on laboratory experience, students are encouraged to engage with practical problem-solving exercises and develop programming skills across a broad range of scenarios.

Keywords: embedded C; teaching; laboratory; PIC; microcontroller

1. Introduction

In modern technology, microprocessors are commonplace. They can be seen in almost any electrical appliance or electronic gadget. Their adoption as a near universal solution in electronic design has come about in part because of their use as miniature computers, but mainly because of their low cost and rapid application development facility.

The dramatic speed at which microprocessor technology has developed is fascinating to observe, but poses a challenge in terms of education. With the technology evolving so rapidly, a course designed for this year's state-of-the-art might be obsolete within a matter of years. Also, the sheer complexity of modern devices begs the question of how much can be covered in a limited period of teaching time. Similarly, the abstraction of complexity in modern electronic devices has led some commentators to speculate if students are exposed to the fundamentals of electronics in the same way as in decades past (von Moller 2010).

This paper discusses the design of a module on microprocessors using the Embedded C language in an Electrical and Electronic Engineering department. The module is aimed at second year undergraduates and uses problem-based learning in the laboratory as the primary means of

*Corresponding author. Email: dlaverty@theiet.org

ISSN 0304-3797 print/ISSN 1469-5898 online

http://dx.doi.org/10.1080/03043797.2012.725711
http://www.tandfonline.com

teaching. The module builds upon teaching in the first year undergraduate modules and focuses on developing the students' understanding of the interface between hardware and software. The students are asked to synthesise knowledge from a variety of previous modules and are given their first experience of developing a 'system'; i.e. a fully working device that achieves a design specification.

The module centres learning on the PIC (originally peripheral interface controller, later rebranded programmable intelligent computer) family of microprocessors. PIC was chosen on the basis of its low cost, flexibility and wide availability. There are plentiful resources for the platform in books, manuals and online programming communities. Development kits are available from a number of manufactures, but the authors felt that they were too sophisticated and abstracted too much of the hardware from the students. A simple development board that keeps the hardware transparent was developed by the authors and is presented in this paper.

This paper describes in detail the structure of the course, including its teaching, laboratories and assessment. Students complete a 'mini-project' as part of the assessment for the course, which is described in detail. Student feedback and evaluation of the course is presented, along with reflections on how the course can be adapted to stay up to date with current practice. It is hoped that this paper can act as a manual for other departments to develop similar Embedded C courses.

2. Related work

There is an acceptance that the use of embedded systems and the ability to program them effectively is of great importance to the engineering field now and in the future (Vahid *et al*. 2008). A course in embedded C programming is considered an excellent point to merge disparate disciplines into a single core module/project (Brylow 2008, Vahid *et al*. 2008). As a result courses in embedded systems programming, usually utilising C as the language of choice, are available from many universities in the UK and worldwide (Colorado 2011, Herriot Watt 2011, IIT Delhi 2011, Leicester 2011). There are also many training solution providers in the area (ENEA 2011, Netrino 2011, TTE Systems 2011), targeting commercial entities with a need to enhance the skills of their staff in the discipline. This complements the various online free educational modules that give a basic overview (EE Herald 2011, IIT Delhi 2011), which does not stray far from the requirements of a basic university course.

The rudimentary structure of an embedded systems course at undergraduate level is given outline in various publications and applied to areas such as operating systems (Brylow 2008), robotics (Passow *et al*. 2011) or C programming (Hovemeyer *et al*. 2011). A major goal of many of these approaches is to produce a hardware implementation that is suitable in terms of cost, simplicity and teaching clarity using on-site university equipment and expertise (Brylow 2008, Chenard *et al*. 2008). Due to regular advances in embedded systems development this leads to specific solutions with specific devices that may become obsolete, unavailable or incompatible with time (Vahid *et al*. 2008). As a result, there is a need to continually assess modern approaches to an embedded systems course until a standard hardware can be defined, should that even come to pass.

The second objective in this area is selection of software and the structure of the laboratories associated with the course. Multiple software solutions have been proposed (Tuma *et al*. 2006, Brylow 2008, Vahid *et al*. 2008, Passow *et al*. 2011). The approach in Passow *et al*. (2011) utilises MPLAB (Microchip 2011) as an effective and low cost resolution. This is also the solution favoured in this paper as an open source, freely available and fairly documented integrated development environment (IDE). Structured laboratories have been previously described in engineering applications (Chang *et al*. 2008); however, in this case the procedure was applied only to an

optoelectronics project and not to software, which has different challenges in teaching and project work.

The third consideration is to determine effective learning objectives and assessment criteria for students. Again, while much excellent work has been done to this end (Rover *et al*. 2008, Passow *et al*. 2011), it has not been appropriately integrated with all other aspects required to establish a rudimentary course from a single source. However the merits of problem-based learning have been shown in Mandri *et al*. (2008) as enhancing the teamwork, knowledge retention and understanding of students in an engineering environment.

Another important factor in producing a high quality course is the selection of teaching environment. Engineers must have a knowledge that goes beyond theory, through experimental work in the laboratory (Feisel *et al*. 2002, 2005). Thus, a laboratory environment is seen as key to effective communication of engineering theory through putting concepts discussed in lectures into practice. Successful operation of undergraduate teaching laboratories is dependent on a number of factors, including the expense of properly maintained equipment and the supply of competent support staff and facilities (Ernst 1983, Weiss *et al*. 2005). A well-implemented laboratory session can stimulate student interest and performance in the subject (Okamura *et al*. 2003).

While considerations have been made regarding teaching and learning objectives and a path to achieve them in academic courses, there has not been a definitive point that focuses on the link between hardware development, software implementation, course content and student evaluation in a laboratory environment. That is the niche that this paper aims to address.

3. Learning objectives

The primary aim of this course is to develop the basic technical skills of the students to prepare them for working with embedded microcontroller systems. This extends from learning the associated technical vocabulary to understanding the many opportunities that exist in the field of embedded systems. Utilising laboratory-based coursework, it is anticipated that the students will develop transferable skills in both software development and hardware implementation (Ernst 1983, Feisel *et al*. 2005), which will prepare the way for further study and ultimately a career in electrical and electronic engineering.

Initially, the course focuses on introducing basic technical skills that allow the students to become sufficiently competent that they can work with minimal supervision. As the course develops, the laboratory work tests the students' problem-solving skills as they endeavour to incorporate new hardware or software designs into their project solutions. Students work in pairs so as to create a sense of community in a positive classroom environment (Davis 1993). A systematic approach to problem solving using pseudo code and flow charts is encouraged. This helps provide clarity for the students by simplifying the problem into smaller, more manageable tasks. Throughout the module, the students are required to maintain a detailed laboratory notebook of their work, which is marked at the end of the module.

A comprehensive list of the learning outcomes has been developed with reference to the classification system suggested by Bloom's Taxonomy (Bloom 1971), and the action verbs suggest by Jackson *et al*. (2003) to define the relevant understanding and to reduce the ambiguity in the specified activity.

By the end of this module, students should be able to:

- apply C programming to embedded systems;
- construct structured C programs using functions;
- demonstrate and recognise the software development processes (compiling and linking) that build executable code from a high level C language source file;

- perform common debugging techniques, such as programming break points, single step code execution, watch/monitor variables, and display variable contents in binary/hexadecimal/decimal;
- describe bitwise C functions and how manipulating digital bits in software can be used for hardware control;
- explain important C data types and the limitations associated with each data type;
- identify and apply casting between data type and how this impacts the value stored in variables;
- discuss program flow control using switch statements and various types of loops;
- recognise, locate and categorise the hardware structure of a microcontroller;
- recognise, locate, categorise, discuss and apply common aspects of embedded microcontroller hardware, specifically: digital input/output, analogue input, analogue-to-digital conversion (ADC), pulse width modulation (PWM), internally generated interrupts for timing and control, externally generated interrupts for program control, interfacing with an external LCD display, serial communications (RS232) for real time data transfer;
- characterise and discuss 'memory mapped' hardware and the Harvard architecture employed by PIC microcontrollers;
- illustrate and demonstrate memory mapped I/O (input/output) as implemented from a high level language such as C.

4. Embedded teaching platform

PIC microcontrollers are popular with both professional electronics developers and amateur hobbyists due to their low cost, wide availability, large user base (which leads to an extensive collection of source code and application note resources online), availability of free development tools and simple programming procedure.

PIC is a family of reduced instruction set computing (RISC) microprocessors made by Microchip Technology (Microchip 2011). Most modern PICs are derived from the PIC1650 (General Instruments 1977), a co-processor designed to handle I/O tasks for the CP1600 CPU. The original PIC stored simple instructions in ROM. All instructions take the same number of cycles to execute and the instruction set is strictly limited. The original instruction set comprised 16 instructions, represented using 4 bits along with operands of size 8 bit. Until very recently (with the introduction of the PIC32 family), PICs employed a Harvard architecture (in which instruction and data memories are separate address spaces) as opposed to von Neumann architectures (in which both instructions and data exist within the same address space).

While the initial specification for the PIC considered its primary purpose to a co-processor for I/O, PICs have proved to be effective microcontrollers for small embedded systems. To increase the versatility of the PIC family, state-of-the-art PICs extend the original memory space and may be clocked at faster rates. Instruction memory is commonly implemented using electrically erasable programmable read-only memory, allowing them to be reprogrammed.

Additionally, PICs are available with a wide range of onboard peripherals (e.g. universal synchronous/asynchronous receiver/transmitters (USART; memory mapped hardware used for serial communications), ADC, PWM motor controllers). Many of these peripherals are taught in this module and are discussed in section 5 of this paper.

4.1. *PIC project board*

The authors have designed a development board for use within this module. Commercial products that do largely the same thing exist, but are often so feature rich that understanding the circuit or

tracing signals through the board is a complex task. The authors desired that students should be able to see and understand the circuit that the PIC was connected to by visual inspection.

A second concern was continuity of device availability. That is to say that the authors did not desire there to be multiple types of development board in use in the same class should some product become unavailable. The development board was designed using CadSoft Eagle Layout Editor (Cadsoft 2011). The embedded platform allows for the following interaction with the microprocessor:

Digital I/O: Analogue:
4 × LED 1 × potentiometer (0–5 V)
3 × push buttons 1 × thermistor (room temperature)
4 × off-board I/O and PWM 4 × external signals (0–5 V)
1 × buzzer

Communications:

RS232/COM port
LCD Interface port

The artwork for the printed circuit board (PCB) is displayed in Figure 1 and a photograph of the component side of the PCB is show in Figure 2. Pdf copies of the PCB are available on request from the primary author.

By inspection of Figure 1, it is possible to see each PCB track on the development board. The circuitry is clearly visible and traceable. Part of the laboratory exercises requires the students to

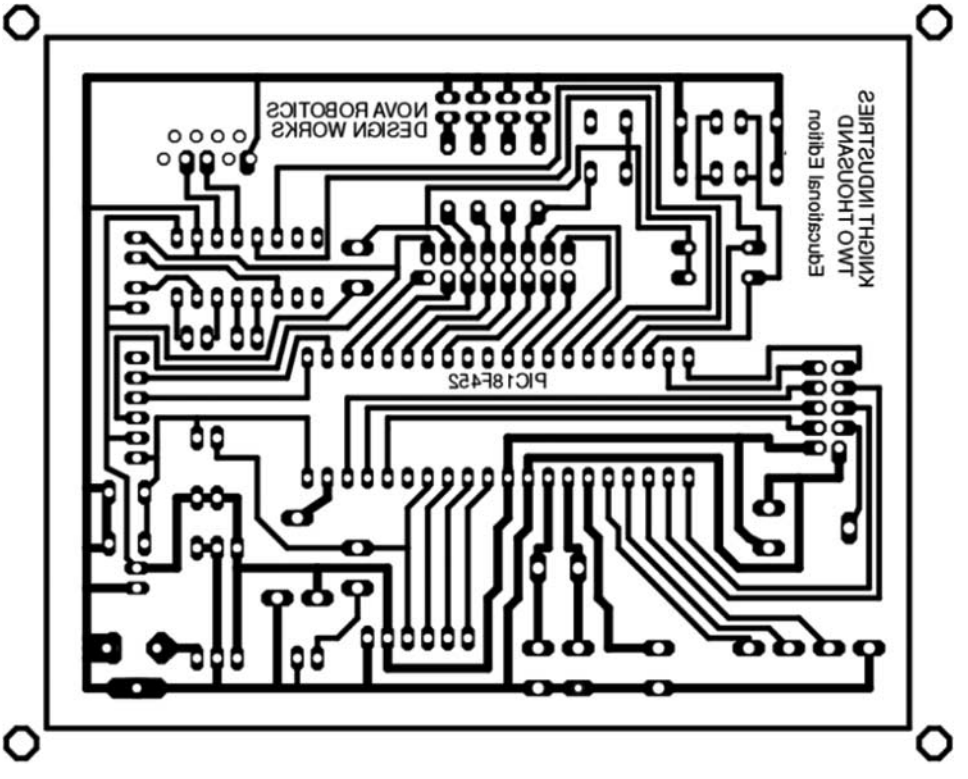


Figure 1. PIC teaching board PCB artwork.

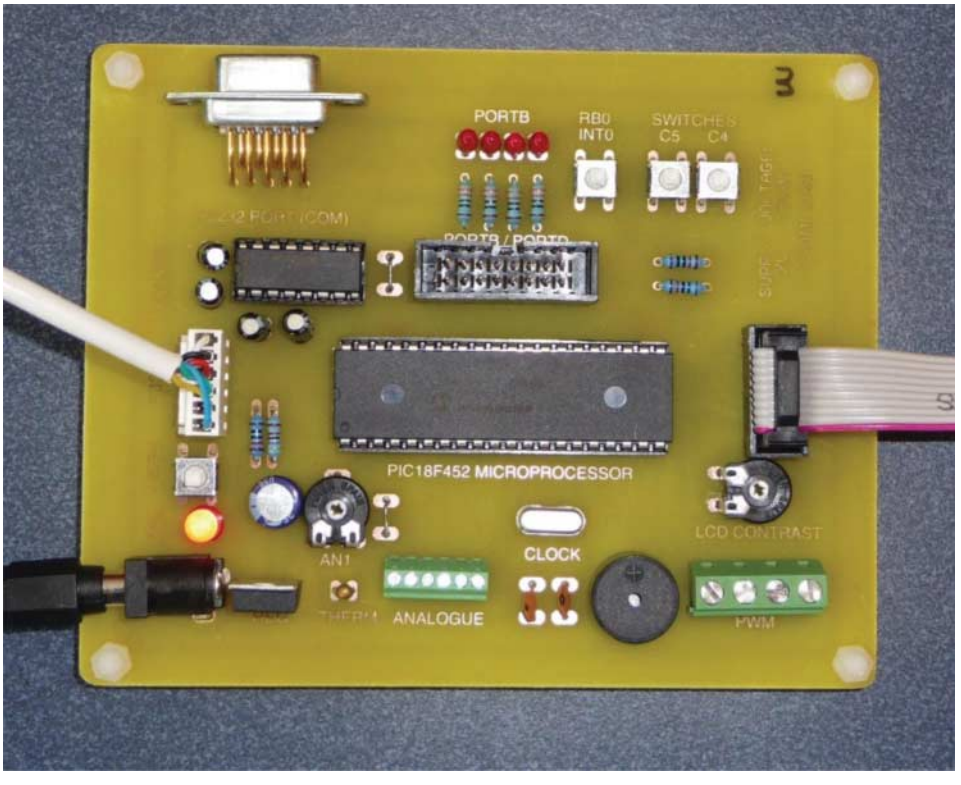


Figure 2. Component side of populated teaching board.

trace the wiring from the push buttons to the microprocessor and use the datasheet to determine the internal connection for programming. In Figure 2, the major components are labelled in copper etched during the PCB fabrication process.

A listing of the components necessary to build the development board is available from the primary author. PCBs were fabricated and assembled within the department by technical staff. The cost for each development board, excluding PCB manufacture and assembly, is circa €25 at September 2011 prices.

4.2. *Workstation configuration*

Each desk in the laboratory (see Figure 3) is equipped with a Windows PC, the 'host', configured with the tools necessary to write, compile and download program code to the PIC microcontroller. Each student writes program code in C in the Microchip MPLAB IDE (Microchip 2011), shown in Figure 4. The IDE compiles the C code into assembly language using the Microchip C18 compiler, which is designed for the 18-series of PIC microcontrollers. The assembly code can be loaded to the microcontroller's program memory.

Configuration of the PIC microcontroller 'target' is achieved using a 'programmer'. In this laboratory, the Microchip ICD2 (in circuit debugger) is used as the programmer. This connects the host via a USB port to the target via the Microchip programming protocol through the target's programming pins (ICD2 socket). Figure 5 shows the implementation process.

A student can make changes to their C code and have it implemented (build and configuration) within the space of a minute, enabling the PIC to be reconfigured several times over the course of the laboratory session.

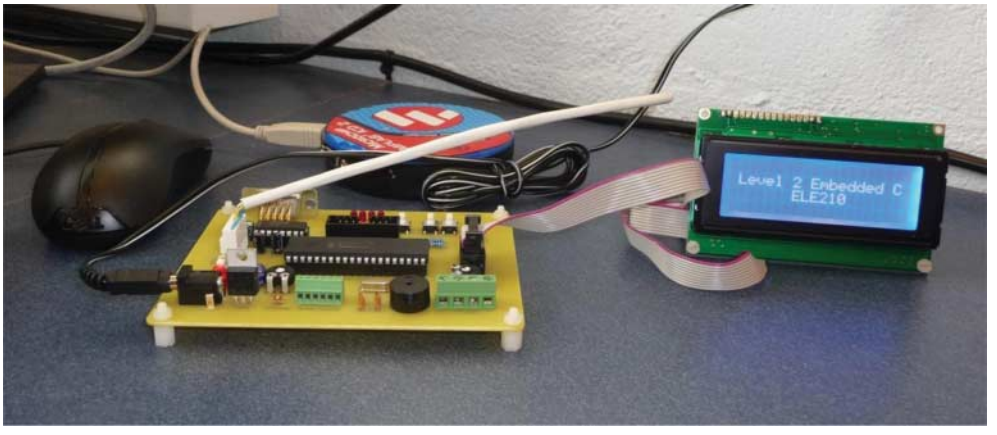


Figure 3. Typical laboratory workstation.

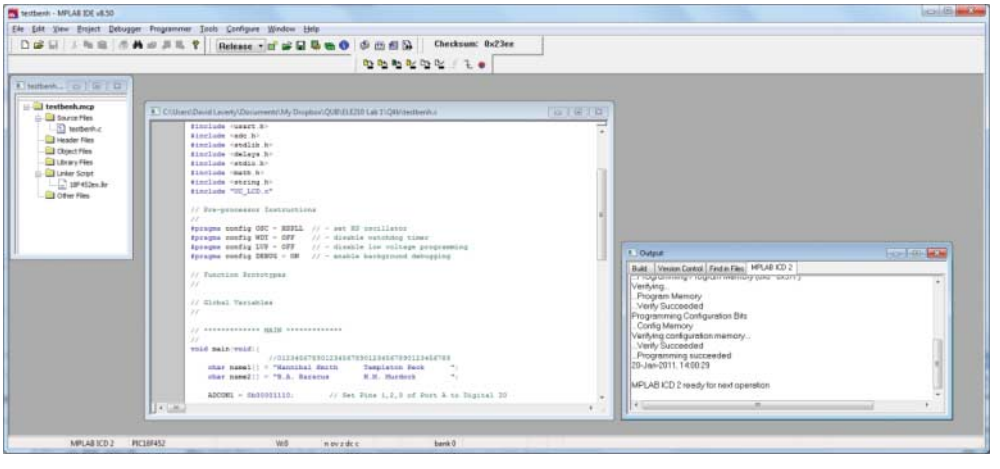


Figure 4. Screenshot of MPLAB IDE.

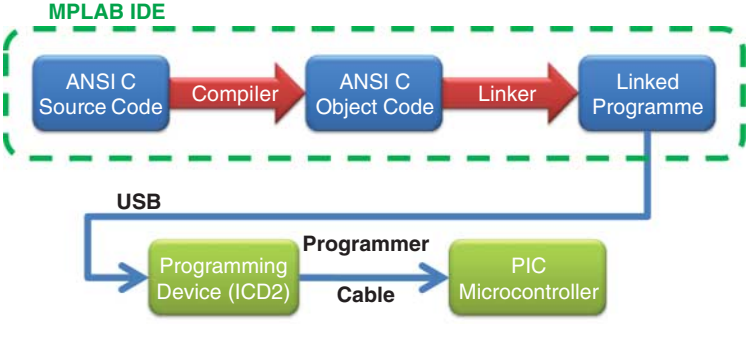


Figure 5. Process of programming PIC microcontroller.

5. Module delivery

The module is composed of a weekly three hour continuous session within the timeframe of one semester. The first hour of the session begins with a lecture, which introduces the problems that

the students will encounter in exercises that day. Any new concepts or hardware that they will be using are discussed and some hints and guidance on how to solve the problems are given. The remaining two hours are conducted in a lab environment. There are nine sessions in total, including two during which students work on a mini-project with minimal demonstrator assistance.

To maintain the focus of the lab session on problem solving, students are given a template that contains the 'boilerplate' code necessary to configure the programming environment for that session. Provision of this code increases the amount of the lab focused on the current learning objective, not disadvantaging students who might not have completed the previous lab and simplifying the role of the demonstrators as all commonly implemented functions are similar. Students are expected to be familiar with how the environment is configured from their first year module on C programming.

Each exercise is broken into 2–3 smaller problems to help guide the students towards completing the exercise. The marks allocated to each step are given and students are made aware of how they will be assessed. Assessment, discussed later, includes preparing pseudo-code in their lab notebook, commenting on the source code and demonstrating that their solution works. Assessment is conducted by a team of demonstrators who are PhD students in relevant fields. Three demonstrators are required for a class of approximate 30 students, in addition to the course tutor.

The students are arranged in pairs so as to encourage teamwork, discussion of the problems and to reduce dependence on help from demonstrators. Literature suggests this can create a sense of community and leads to a positive classroom environment (Davis 1993). Although there was concern that it would be difficult to differentiate students when they work on a problem in pairs, the subsequent assessment of the lab notebooks provided sufficient material on which to individualise assessment.

The class size for this module is typically 28–29 students. Cohort statistics are given in Table 1. In 2011, the class size was 16 due to a poor intake of students the previous year. As this is a second year undergraduate module, the mean age of students is 19–20 years, with occasional mature students. The gender balance is typical of the electrical engineering pathway in the university, usually 92–97% male.

5.1. *Laboratory sessions*

There are seven taught laboratory sessions, followed by two sessions during which students work on a mini-project. This section outlines the content presented in the taught laboratory sessions. The laboratory session titles, hours and percentage of marks are presented in Table 2.

5.1.1. *Session 1: digital input and output*

This introductory session was designed to familiarise the students with the hardware and software resources at their disposal. Students write a simple program to turn on LEDs and read the status

Table 1. Module cohort statistics

Year	Class size	Typical age (years)	Gender ratio (male:female)
2008	28	19–20	26:2
2009	28	19–20	27:1
2010	29	19–20	27:2
2011	16	19–20	15:1

Table 2. Course overview

Session	Content	Time allocated	% of total mark
1	Digital input & output	2 hours	5
2	LCD display	2 hours	5
3	Advanced LCD display and functions	2 hours	5
4	Analogue input	2 hours	5
5	Interrupts and dimmer switch	2 hours	5
6	Timers	2 hours	5
7	RS232, read & write via the COM port	2 hours	5
8	GPS project	4 hours	5
9			
Total		18 hours	40%

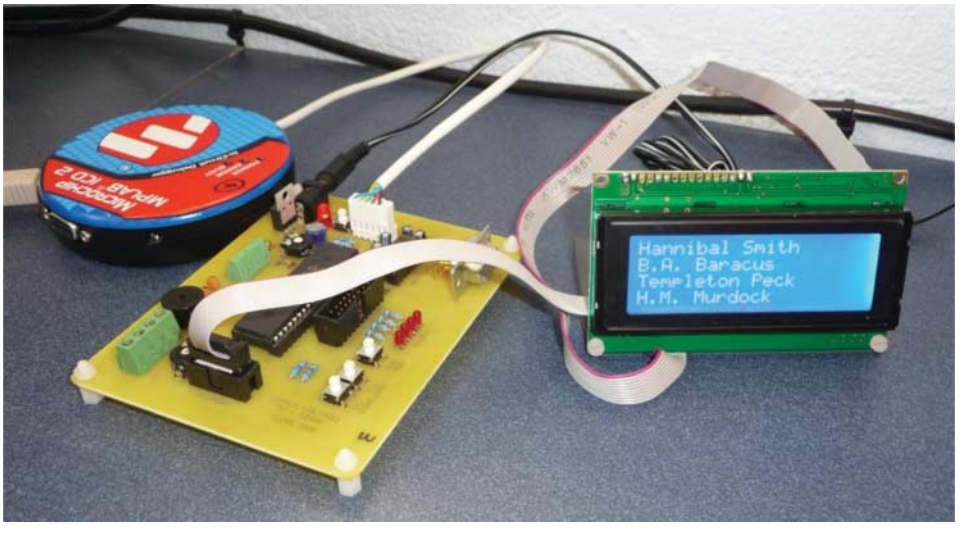


Figure 6. Students' names printed to the LCD screen.

of push buttons. The advanced level involves writing a program to produce a scrolling pattern on the LEDs and controlling the speed via push buttons.

5.1.2. *Session 2: LCD display*

The students are provided with source code to operate an LCD character display. In the first problem, students must print their name to the display, as in Figure 6. In the second problem, students reproduce the scrolling LED effect from the previous week using an asterisk character on the LCD screen.

5.1.3. *Session 3: advanced LCD display functions*

In week three, students develop a program to display numerical data on the LCD screen. This involves converting an integer to an ASCII character string and then printing this to a display. The advanced problem introduces the case of negative numbers and in later weeks students will refer back to this code when they are asked to print floating point numbers to the LCD screen.

5.1.4. *Session 4: analogue input*

Week four introduces ADC. For many students, this will be their first hands-on experience with an ADC and they will be required to consider the numerical representation of real-world signals. The first problem asks students to read the value of the potentiometer and display the raw output of the 10-bit ADC (0–1023), subsequently scaling this to 0–5 V. They must show that the value on the LCD changes as they adjust the potentiometer.

In the second part of this lab, students use the thermistor to create a room temperature thermometer. They are supplied with a lab thermocouple and must devise a method of displaying the true temperature of the thermistor in degrees Celsius on the LCD screen.

5.1.5. *Session 5: timers*

The fifth lab introduces the microprocessor's timers. In the first exercise, students must code a clock that displays the time since 'boot-up' in hours, minutes and seconds on the LCD screen. This involves determining correct settings for the clock multiplier and the timer dividers.

The second exercise is to code a stopwatch. This displays the time between button presses in hours, minutes, seconds and milliseconds. Introducing more complex digital I/O problems, students are challenged to use one button to start/stop the clock, reset if held for more than three seconds, and use a second button to display the 'split time', i.e. the time at the moment the button is pressed, for four seconds before resuming normal operation.

5.1.6. *Session 6: interrupts and dimmer switch*

In the first exercise, students are asked to rewrite the stopwatch program from the previous week's lab using interrupts triggered by the push buttons as opposed to polling the state of the buttons. Demonstrators are required to explain the advantages of this method when used in more complex applications.

The second exercise uses a timer interrupt to create a 'dimmer switch' by programming a PWM waveform of frequency 50 Hz with a duty cycle modulated by the potentiometer introduced in week four. Turning the potentiometer up and down varies the brightness of the LEDs. Students can easily relate to how this code is useful in a real-world application.

5.1.7. *Session 7: serial communications (RS232/COM port)*

In the final session before the mini-project, students are introduced to the idea of reading data from and writing data to the microprocessor by means of serial communications. In the first exercise, students write an asterisk character using the onboard USART to the RS232 port of the microprocessor each time a push button is pressed. Correct operation is confirmed by observing the connection on a terminal emulator program on the PC, such as Windows HyperTerminal. The exercise then asks for the ADC value to be printed to the RS232 port once per second.

Exercise two, reading data from the RS232 port, asks the students to code a clock such that the present date and time may be configured from the PC terminal emulator. Subsequently, an alarm clock that will trigger the buzzer on the development board at a time set via RS232 is developed. The alarm is reset by a digital interrupt. The final problem is to add a function so that a message typed on the PC terminal emulator is displayed on the LCD screen.

This session is very demanding but the authors have been impressed each year that students are seemingly adept at creating new solutions to these problems. Specifically, in this exercise, students must devise methods of buffering and parsing data, a new concept within the syllabus at this level.

5.2. *Mini-project*

The mini-project is designed to evaluate how well students have understood the programming techniques they developed in the lab sessions through the application of that knowledge to a new, more demanding problem. The mini-project is a significant part of their assessment and independent working without demonstrator involvement is encouraged.

The project that has been used each year this module has operated is to develop a GPS satellite navigation system. Students are presented with a project sheet outlining a problem specification, which states system operation, necessary background information and a few suggestions on a strategy for getting started. There is no 'correct' solution; rather marks are be awarded for meeting the specification, quality of code design, well-commented source code, written explanation of code in the lab notebook and creativity.

The project requires students to receive, parse and utilise data from a GPS receiver. The provided GPS receiver outputs data in National Marine Electronics Association (NMEA) (NMEA 2011), format via RS232. The project sheet gives a specification for the NMEA sentences that students will need to complete the project. These include:

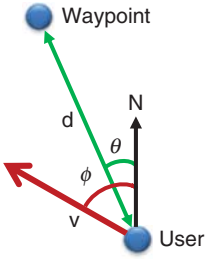
- recommended minimum data for GPS (RMC);
- Bearing and distance to waypoint using a great circle route (BWC);
- Global positioning system fix data (GGA);
- Satellite status and dilution of precision (GSA).

NMEA sentences are variable length, comma separated and parsable ASCII strings, terminated with a checksum. Some code examples are available on the Internet for decoding NMEA and students are encouraged to examine these to help with their own solution. They should document this research in their lab notebooks.

For full credit in the project, the specification requires that the system shall:

- identify and parse sentences according to the type identifier;
- display on the LCD screen:
  - — time and date (compensating for daylight savings);
  - — latitude;
  - — longitude;
  - — altitude;
- evaluate the checksum;
- calculate and display estimated time of arrival (ETA) at next waypoint based on speed and bearing;
- graphically display proximity to waypoint on LCD screen.

Achieving a complete solution to the mini-project requires that the student demonstrates an ability to think logically about several programming challenges. The systems require the ability to receive long strings of data by RS232 at high baud-rate, parse the data using tokens, enumerate the data, apply trigonometric functions and work with time-of-day (hh:mm:ss) representations of numbers. Figure 7 shows a diagram used to illustrate to students the method in calculating the ETA. Figure 8 shows a complete, working, example solution to the mini-project.



Parameters 'd', 'v', ' ϕ ',and 'θ ' are available from NMEA sentences.

Resolve speed 'v' in direction of Waypoint and calculate estimated time of arrival (ETA) based on distance 'd' and time of day from NMEA sentences.

Figure 7. Diagram of ETA problem in mini-project.

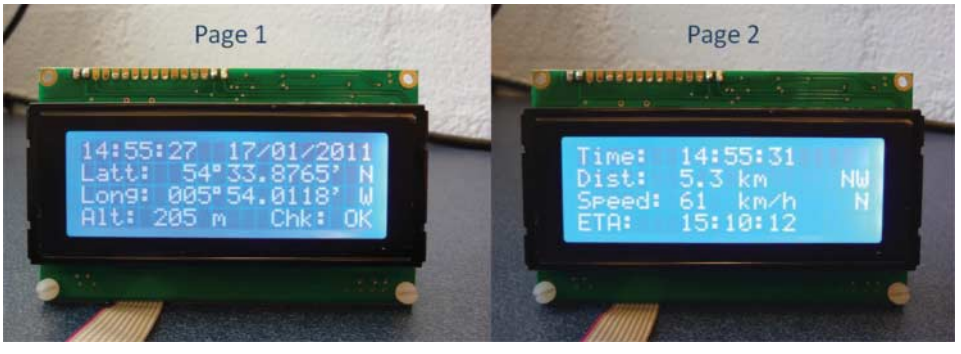


Figure 8. Photograph of complete working GPS 'Sat-Nav' display.

6. Assessment and feedback

This type of lab-based problem-solving course lends itself to continuous assessment by virtue of the exercises that the students complete in each weekly session. However, given that students work in pairs and are encouraged to discuss ideas with their classmates, the authors felt that continuous assessment alone would be insufficient to differentiate the more able students from their cohort. Consequently, the students' final assessment is composed of their lab marks, a mark awarded for their lab notebook, and a mark awarded for completeness of the mini-project and a verbal discussion of the mini-project to assess their understanding of their own solution. As explained to students at the beginning of the module, and given in the rubric for assessment, marks are allocated as follows: 40% for continuous assessment of lab exercises; 40% for laboratory notebooks; 20% for the 'mini-project' assignment. The methods by which each component is assessed are now discussed.

6.1. *Continuous assessment*

Each lab exercise is marked according to four parameters. Students are made aware of the marking scheme for the labs at the start of the module and it is continually reinforced by demonstrators. The four parameters used are as follows:

(1) Description of problem/pseudo code in notebooks.
(2) Quality of source code comments.
(3) Completeness of solution.
(4) Understanding of solution.

Marking the description of the problem, pseudo-code and code comments is subjective, so demonstrators are asked to confer with each other to ensure that standard marks are awarded across the class. It is important to insure that inter-rater and intra-rater scoring is consistent (Center for Advanced Research on Language Acquisition 2011). A short rubric is presented for demonstrators to guide course assessment, with further fine-grain comparative ranking performed in order to differentiate between students who exceed expectations and those who achieve the bare minimum for a rubric grade. Since demonstrators move about the laboratory through the session, often different parts of the exercises will be marked by different demonstrators. Demonstrators have been successful in delivering consistent assessment during the years the course has operated.

The completeness of the solution is marked strictly in accordance with the problem specification, but the students have an opportunity to improve their marks if they can verbally demonstrate their understanding of the problem and present pseudo-code that they would implement given more time.

6.2. *Lab notebooks and mini-project*

Towards the end of the semester and after the completion of the mini-project, students submit their lab notebooks for assessment. Notebooks are assessed based on the students' ability to replicate the work that was taught during the laboratory session. Assessment is primarily based on logical organisation and that important observations are recorded; however, a small amount of marks are awarded for neatness and quality of writing and diagrams.

6.3. *Mini-project*

Throughout the course, students are encouraged to collaborate and share ideas. In the mini-project, students are informed that, while collaboration is still encouraged, plagiarism of code or sharing substantial code segments is not allowed and will be marked down accordingly if discovered. To ensure that ideas are original and well understood, each student has a one-to-one interview with a demonstrator lasting approximately 10 minutes. The demonstrator will ask a number of questions from a prepared set provided from the module coordinator. In the interview, the demonstrator will ask to see the mini-project in operation and will ask the student to explain a particular piece of their source code.

The mini-project is also marked in terms of completeness of the project specification and, in order to allow the more able students to be differentiated, the most challenging aspects of the specification are marked strictly. Even so, each year several students are able to complete the most difficult problems, sometimes in unique and creative ways.

6.4. *Results*

The results from the three years that the course has operated are given in Figure 9. Enrolment in the module was 28 students in 2009, 29 in 2010 and 16 in 2011. Results for the module are generally high, tending towards classifications at first class ($>70\%$) or 2:1 ($>60\%$). The class of 2010 was a particularly able cohort, with 72% of the students passing the module with a final result above 70%. Of those students who failed the course ($<40\%$), most had poor attendance and had effectively dropped the module. Class marks have remained generally consistent through each year.

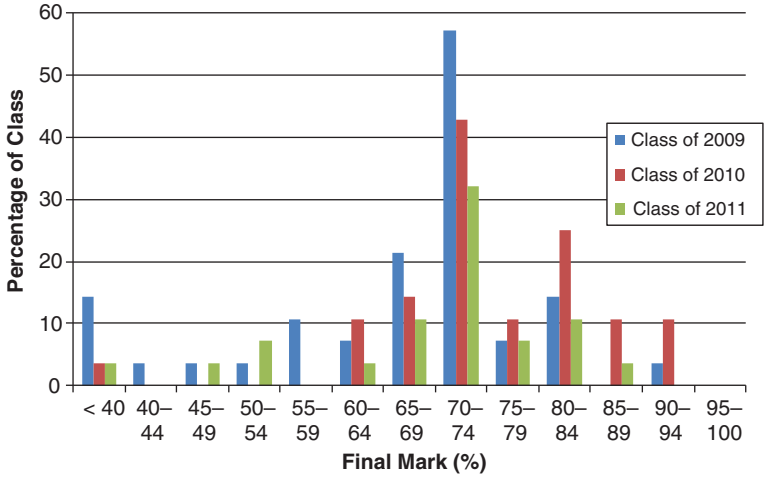


Figure 9. Distribution of final marks for the classes of 2009 to 2011.

6.5. *Feedback*

Students are given informal feedback at each stage of assessment during their laboratory sessions with a view of helping them achieve maximum performance and benefit their learning. Many students have been observed to benefit from this and indeed begin to pass on help and suggestions to their colleagues.

Since there are no formal reports submitted during this module, that avenue for providing traditional feedback is not available. Instead, students receive more direct feedback during the interview when they demonstrate their mini-project. Demonstrators will have been prepared with suggestions and recommendations to aid students with future programming-related projects to assist their further study on the course. Demonstrators are instructed to be constructive and positive.

7. Student reception

Student feedback is obtained as standard procedure for all modules in the department through the Teaching Evaluation Questionnaire (TEQ) distributed at the end of teaching in each subject. In addition to this, a questionnaire was sent by email to students regarding this particular module. Students were asked to comment on the module in terms of delivery, assessment, content and the quality of demonstration and teaching they received.

The TEQ asks for students to rate the module on a scale of 1 to 5 over a range of criteria. Results from the TEQs were consistent over the three years the course has run, consistently scoring an average above 4 in terms of its aims, purpose and quality of teaching. Similarly, it scored highly in terms of interest, quality of laboratory equipment, organisation and challenging subject matter. Scores dipped slightly, indicating a known issue with the quality of the computers in the laboratories, and student response has proven useful in structuring demonstrator training to improve interaction with students. From this evidence, it is clear that adequate computing facilities and well-trained demonstrators are an important resource for student satisfaction and performance.

Evaluating the performance of the module from the perception of students also allows appraisal of qualitative educational performance. Considering these data, the consensus is that the module has a rigorous workload but the learning curve is sufficient for all student abilities. External

validation of the merits of the course comes from two sources: the consistent positive feedback garnered from student assessments; the acceptance of the module by outside accrediting bodies.

7.1. *Questionnaire*

The response from the email questionnaire expands on the detail provided by the TEQ and is provided by section.

7.1.1. *Module delivery and assessment*

Students appreciated the laboratory-based delivery and assessment of the module, as opposed to traditional lecture and written exam-orientated courses. The response indicates that students prefer each week being taught as a set of programming exercises rather than a sit-down lecture. They also prefer this approach to other labs, where step-by-step instructions are given to perform an experiment, requiring little thought or insight into the subject.

Some respondents commented regarding the pace of the course, stating that they felt it was one of the more demanding modules in terms of how quickly new themes were introduced. Although none was directly critical of this, the authors feel it is worth keeping in mind and perhaps could be addressed by slightly restructuring the laboratory exercises set.

7.1.2. *Content*

Students were happy with the amount of material covered throughout the module and the pace at which the content was taught. The authors assert this validates the approach of the module, as ensuring a good balance between challenging content and sufficient learning time was of paramount concern from course inception.

7.1.3. *Demonstrators*

Three to four PhD students in relevant subject areas were required to give adequate assistance to a class of 30 students, taking into account variable ability within the class. This equates to a demonstrator to student ratio of approximately 1:10.

Overall, students seemed satisfied with the provision of demonstrators during the delivery of the course, with only a few students mentioning that they would have preferred more time with demonstrators. It is a difficult balance to strike, as independent thought on the problems is vital for learning, but the nature of some of the exercises mean that many solutions are possible and expert help is needed to provide an environment for discussing methods of problem solving.

7.1.4. *Other comments*

Some respondents felt that they would benefit from having more time to work on the exercises outside of laboratory time, rather than attempting to solve all the problems in a single session. Given that the teaching boards, programming equipment and software are all low cost, this could be facilitated in future years but would require an adjustment in the methods of assessment. The authors are concerned that this could lead to greater scope for students to share or plagiarise code from one another.

Interest was also expressed in a follow-up course focusing on interfacing with other technologies, such as keypads and card-swipes. Since the module effectively teaches the concepts

of embedded software design (e.g. using interrupts, timers, serial communications), the skills learned could be applied to teach other courses (e.g. control systems, digital signal processing (DSP), power and energy systems), where interfacing with hardware/software systems is necessary.

Some students who are now employed full time in embedded programming have provided feedback, saying that the module was of excellent benefit for them as a foundation for their new careers. Many students have recommended the module to new students beginning their second year of study.

7.2. *External assessment*

Evaluation of the performance of the module from examination results and student satisfaction is complemented by the continued support of positive reviews from external examiners each year. Additionally, in the timeframe in which the module has been running, the course has successfully had Institution of Engineering and Technology accreditation renewed, further emphasising the high quality of the content and delivery method.

8. Conclusions

This paper has introduced a new Embedded C course, designed to develop students' understanding of the interface between hardware and software in electronic systems through problem-based learning. The course structure, content and delivery are described and the relevant teaching materials including the microprocessor development board are described and available on request from the authors.

The structure and content of the laboratory sessions has been described in detail, describing how the course builds students' knowledge and experience of Embedded C code through problem-based exercises. Real-world examples of problems are used in some exercises so that students can relate the course content to situations that they have prior experience with. The course culminates in a mini-project, which gives students the opportunity to apply what they have studied in a less directed setting.

The course delivers content and new themes at a high pace, yet students with a range of abilities have demonstrated they can stay apace with the class and retain new content and ideas each week through their solutions to the laboratory exercises. Concerns regarding individually assessing students through group work were successfully mitigated through creating opportunities for the more able student to differentiate themselves through project assessment and notebook assessment.

Overall, students have been engaged with the module, worked diligently at their projects and have provided the authors with positive comments and constructive criticism. Many students have gone on to recommend the module to others and credit the module for providing them with a foundation in their new careers in embedded programming.

References

Bloom, B.S., 1971. *Taxonomy of educational objectives: The classification of educational goals. Handbook 1, cognitive domain*. London: Longman Group; Michigan, Edwards Bros.

Brylow, D., 2008. *An experimental laboratory environment for teaching embedded operating systems*. ACM SIGCSE Bulletin – SIGCSE 08, New York: ACM SIGCSE.

Cadsoft, 2011. *Eagle product information* [online]. Cadsoft Computers. Available from: http://www.cadsoft.de/freeware.htm [Accessed 19 January 2010].

Center for Advanced Research on Language Acquisition, 2011. *Evaluation* [online]. University of Minnesota. Available from: http://www.carla.umn.edu/assessment/vac/evaluation/p_2.html [Accessed 8 June 2012].
Chang, G.-W. *et al.*, 2008. A progressive design approach to enhance project-based learning in applied electronics through an optoelectronic sensing project. *IEEE Transactions on Education*, 51 (2), 220–223.
Chenard, J.-S. *et al.*, 2008. A laboratory setup and teaching methodology for wireless and mobile embedded systems. *IEEE Transactions on Education*, 51 (3), 378–384.
Davis, B.G., 1993. *Tools for teaching*. San Francisco: Jossey-Bass.
EE Herald, 2011. *Course on embedded systems* [online]. Available from: http://www.eeherald.com/section/design-guide/esmod1.html [Accessed 24 June 2011].
ENEA, 2011. *C for embedded systems* [online]. Available from: http://www.enea.com/templates/CourseDescriptionPage____19105.aspx [Accessed 24 June 2011].
Ernst, E.W., 1983. A new role for the undergraduate engineering laboratory. *IEEE Transactions on Education*, 26 (2), 49–51.
Feisel, L.D. *et al.*, 2002. The challenge of the laboratory in engineering education. *Journal of Engineering Education*, 91 (4), 367–368.
Feisel, L.D. *et al.*, 2005. The role of the laboratory in undergraduate engineering education. *Journal of Engineering Education*, 94 (1), 121–130.
General Instruments, 1977. *Data catalog: micro electronics from General Instrument Corporation* [online]. Available from: http://www.rhoent.com/picsw.pdf [Accessed 18 January 2011].
Hovemeyer, D., *et al.*, 2011. Teaching embedded systems using AVR microcontrollers. *Journal of Computing Science in Colleges*, 26 (3), 104–105.
IIT Delhi, 2011. *Free video lectures, embedded systems* [online]. Available from: http://freevideolectures.com/Course/2341/Embedded-Systems [Accessed 24 June 2011].
Jackson, N., Wisdom, J. and Shaw, M., 2003. *Guide for busy academics using learning outcomes to design a course and assess learning* [online]. LTSN Generic Centre. Available from: http://bit.ly/ADfZHE [Accessed January 2012].
Mantri, A., Dutt, S., Gupta, J.P. and Chitkara, M., 2008. Design and evaluation of a PBL-based course in analog electronics. *IEEE Transactions on Education*, 51 (4), 432–438.
Microchip Technology Inc., 2011. *MPLAB integrated development environment* [online]. Available from: http://www.microchip.com/ [Accessed 18 January 2011].
NMEA, 2011. *NMEA 0183* [online]. Available from: http://www.nmea.org/ [Accessed 18 January 2011].
Netrino, 2011. *Embedded software training calendar* [online]. Available from: http://www.netrino.com/Embedded-Systems/Training-Courses/Calendar [Accessed 24 June 2011].
Okamura, A.M. et al., 2003. Feeling is believing: using a force-feedback joystick to teach dynamic systems. *Journal of Engineering Education*, 91 (3), 345–349.
Passow, B.N. *et al.*, 2011. *An open platform for teaching and project based work at the undergraduate and postgraduate level* [online]. De Montfort University Open Research Archive. Available from: https://www.dora.dmu.ac.uk/xmlui/handle/2086/4507 [Accessed 24 June 2011].
Rover, D.T. *et al.*, 2008. Reflections on teaching and learning in an advanced undergraduate course in embedded systems. *IEEE Transactions on Education*, 51 (3), 400–412.
TTE Systems, 2011. *Prog. Tech. for reliable embedded systems* (*C*) [online]. Available from: http://www.tte-systems.com/services/training/intro [Accessed 24 June 2011].
Tuma, T. and Fajfar, I., 2006. A new curriculum for teaching embedded systems at the University of Ljubljana. *In: Proceedings of the 7th International Conference on Information Technology-Based Higher Education Training*. Piscataway, NJ: IEEE, 14–19.
University of Colorado, 2011. *Embedded system design* [online]. Available from: http://ecee.colorado.edu/~mcclurel/syllabus.html [Accessed 24 June 2011].
University of Herriot Watt, 2011. *Embedded systems MSc/Diploma* [online]. Available from: http://www.postgraduate.hw.ac.uk/course/64/ [Accessed 24 June 2011].
University of Leicester, 2011. *Programming embedded systems* [online]. Available from: http://www.le.ac.uk/eg/mjp9/pttesguide.htm [Accessed 24 June 2011].
Vahid, F. *et al.*, 2008. *Timing is everything – embedded systems demand early teaching of structured time-oriented programming*. WESE 2008, Atlanta GA.
von Moller, K., 2010. *State of electronics* [online]. Available from: http://www.karlvonmoller.com/blog/2010/07/25/state-of-electronics-the-beginning/ [Accessed 17 January 2011].
Weiss, B., Grindling, G. and Proske, M., 2005. A case study in efficient microcontroller education. *ACM SIGBED Review*, 2 (4), 36–43.

About the authors

David M. Laverty received MEng and PhD degrees from Queen's University Belfast, Belfast, UK, in 2006 and 2010 respectively. Since graduating he has been with the Energy, Power and Intelligent Control (EPIC) Cluster at Queen's University Belfast, Belfast, UK. He lectures Smart Grid and Energy Systems and has research interests including real-time monitoring and wireless telecommunications in the utility environment. Dr Laverty is a member of, and volunteers with, the IEEE and the Institution of Engineering and Technology.

Jonny Milliken graduated in 2009 with an MEng in Electrical & Electronic Engineering from Queens University Belfast (QUB). His dissertation studied Wireless Intrusion Detection and established the foundations of a subsequent PhD investigating Cross Layer Techniques for Intrusion Tolerant Network Design at QUB. Jonny is also a member of IEEE and involved with IAESTE in Northern Ireland.

Matthew Milford graduated in 2009 with an MEng in Electronic and Software Engineering from Queen's University Belfast. Matthew's dissertation focussed on Soft Processor Design for FPGA, which forms the basis of a PhD investigating Compiler Design for Embedded Streaming Systems. Matthew is also an active member of IET Northern Ireland. Matthew is a member of the IEEE.

Michael Cregan graduated from The University of Ulster in 1990 with an Honours Degree in Engineering and from Queeen's University Belfast with an MSc and a PhD. He is currently working at Queen's University Belfast. Prior to this he has worked for Northern Ireland Electricity and General Controls and Automation.