USE OF FreeRTOS IN TEACHING A REAL-TIME EMBEDDED SYSTEMS DESIGN COURSE

Nannan He, Han-Way Huang
Electrical and Computer Engineering Technology Department
Minnesota State University at Mankato

Abstract

We have recently made use of a free and open source Real-Time Operating System (RTOS) called FreeRTOS in teaching engineering students the real-time embedded systems design course. Different from many existing real-time computing courses, we focus on introducing students to the design and application development of real-time embedded systems from the practitioner's point of view, instead of introducing research or theoretical topics. FreeRTOS is a real-time kernel designed to run on a microcontroller for embedded applications. It supports a large number of microcontroller architectures and has become the leading real-time computing platform. In this course, we have applied this real-world RTOS in both lectures and lab sessions, from the case study for studying basic RTOS concepts, conducting lab experiments with multitask scheduling, resource and service management and real-time interfacing with microcontrollers, to developing capstone projects. Our teaching experiences demonstrate that FreeRTOS is a richly featured, cost-efficient and open source software platform to teach engineering students the real-time embedded systems design and the development of microcontroller-based real-time applications.

Introduction

Nowadays, with the emergence of new processors and methods of processing, communications and infrastructure, modern industrial automation systems require thigh performance and real-time control capabilities. There is an on-going effort to achieve the education goal of increasing the technical depth and broaden training by investigating deterministic timing techniques for complex real-time automation systems in our department. As an important exploration step towards this goal, a new real-time embedded system design course has been offered to electrical engineering and computer engineering senior and graduate level students. At the same time, the goal is also an important guideline in the course preparation and teaching practices, as a result some special features of this course are formed compared with many existing real-time systems computing courses.

Real-time computing courses have been widely offered to computer science major students at the senior or graduate level, with the aim of equipping them with the knowledge of conducting scientific research in this area. These courses typically focus on the theoretic topics in real-time computing, such as various formal models of real-time systems, schedulability theory, design and timing analysis of multi-task scheduling algorithms and operating systems. Distinguished from these courses, the real-time embedded systems design course presented in this paper emphasizes engineering issues of designing and developing real-time systems in practical embedded applications. The course is taken by senior electrical engineering and computer engineering students and some graduate students pursuing their Master degree. Other engineering students with the appropriate software background could also take the course. It focuses on teaching engineering students real-time systems design and applications from the practitioner's point of view, instead of targeting research and theoretical topics as conventional real-time computing courses. After taking this course, students are expected to demonstrate the ability to correctly define and design real-time systems, and the ability of basic real-time application development. Active learning and hands-on learning are fundamental teaching

approaches applied to this real-time systems design course.

This new course covers the topics on RTOS relevant topics such as multi-task scheduling, system services, and resource policies, and some application issues for developing real-time systems such as microcontroller, requirement analysis, performance analysis and verification of real-time system design. Among these topics, RTOS is one of the core components of our course. In this course, we employed the free, open source software called FreeRTOS as a case study of teaching RTOS in both lectures and lab sessions.

FreeRTOS is a full-featured real-time kernel/scheduler designed to be small enough to run on a microcontroller. It provides the real time scheduling functionality, inter-task communication, timing analysis and synchronization primitives. It also offers rich example projects as the basis for developing embedded real-time systems. Moreover, this software supports a large number of underlying microcontroller architectures including the advanced ARM Cortex™-Mx series, and has become the standard RTOS for developing microcontroller-based applications. To simplify the complexity of the application code, FreeRTOS provides a rich set of time-related Application Programming Interfaces (APIs). As a result, complex embedded real-time applications can be efficiently constructed to meet their real-time processing deadlines. In this course, FreeRTOS has been applied to conducting experiments with multitask scheduling algorithms and real-time interfacing with microcontrollers for all our lab sessions and course projects.

This paper presents our primary experiences in teaching real-time embedded systems design to engineering students, with the emphasis on how we adopted FreeRTOS, as a real-world RTOS example in this course, to improve teaching effectiveness. The description of this course is first given, including course contents, learning outcomes and instructional approach. Next, a survey of existing RTOSs is reported. Then, this paper presents the software FreeRTOS and how we have made use of FreeRTOS in lab assignments and course projects from exercises preparation, lab setup and organization. Finally, the paper gives a conclusion and discusses future work.

Course description

Our new real-time embedded systems design course targets the learning of real-time systems design and applications from the practitioner's point of view. It has been offered for two years. This course is organized as two hours of lecture and three hours of laboratory per week. It has three main objectives.

- To improve students' awareness of real-time specifications in critical automation controllers and other embedded systems.
- For engineering students to apply modern development tools and advanced techniques to designing and analyzing the performance of small-scale real-time systems.
- To enable students to develop real-time applications to solve problems with specific timing requirements.

In order to accomplish the basic instruction approach of active learning and hands-on learning, this course has an experiential component which is the most important. It allows students to apply advanced techniques learned in this course to develop an understanding of their advantages and disadvantages in different applications.

Course learning outcomes

Our overall aim is to equip students with the knowledge of designing real-time systems and developing real-time applications to solve engineering problems in practice. In the development of this course, we identify course learning outcomes that stem from this aim and extend to learning activities. We developed

twelve learning outcomes, classified into four core components as shown in Figure 1.

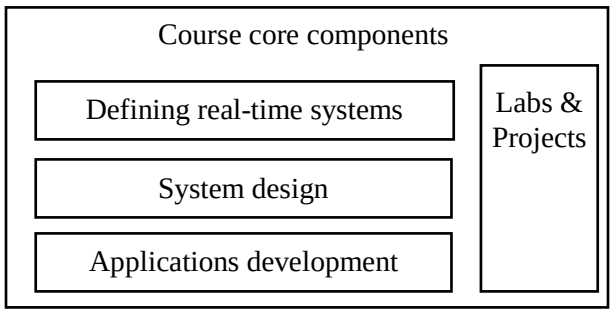

Figure 1: Core components of the course.

1. To demonstrate the ability to correctly define real-time systems

   1.1 To identify problems as hard, firm or soft real-time system;

   1.2 To articulate and compare different definitions in real-time systems.

2. To demonstrate the ability of real-time systems design

   2.1 To comprehend formal specifications based approaches and utilize formal modeling tools;

   2.2 To understand the impact of hardware on real-time performance;

   2.3 To analyze the scheduling feasibility of a set of independent tasks and derive schedules;

   2.4 To understand resource policies and system services for inter-task communications and synchronization;

   2.5 To understand the challenges and applications of performance analysis techniques;

   2.6 To understand real-time issues on advanced distributed control networks such as SCADA;

3. To demonstrate the ability of basic real-time application development

   3.1 To understand real-time software testing, verification and system integration.

   3.2 To be aware of performance optimization techniques.

   3.3 To utilize modern tools to simulate executions and critique different implementation choices.

   3.4 To comprehend the architectures, functions and applications of existing RTOSs.

Course Contents

The topics covered in this course include real-time scheduling approaches such as clock-driven scheduling, static and dynamic priority driven scheduling, resource handling, timing analysis, RTOS, hard and soft real-time systems, distributed real-time systems, concepts involved in the modeling, design, analysis and verification of real-time systems. Course materials were drawn from two text books [1, 2], FreeRTOS tutorial book and reference manual [3], ARM Cortex-M microcontrollers' datasheets, websites, and other publications. Table 1 shows the organization of these topics. In this table, the time schedule of each top-level topic is indicated for this one semester course (~ 15 weeks). Please note that topic 7 - *Case studies (FreeRTOS)* is not labeled with a specific schedule because its sub topics are provided in combination with other topics throughout the semester.

Instruction Approach

Active learning and hands-on learning are fundamental teaching approaches applied to this real-time system design course. All of the classes were held in the laboratory. For this course, this setting eases the class discussion and flexible adoption of a variety of teaching methods, depending on the characteristics of the

Table 1: Course Topics.

1. Fundamentals (week 1)
Basic concepts and misconceptions
Multidisciplinary design challenges
2. Hardware for real-time systems (2^{nd}-3^{rd})
Processor architecture
Architectural advancements
Peripheral interfacing
Distributed real-time architecture
3. Real-time operating systems (3^{rd} - 6^{th})
Multi-task scheduling
System services for application programs
RTOSs selection issues
4. Requirement engineering (6^{th} – 8^{th})
Requirement Eng. for real-time systems
(semi-) Formal methods in system specification
Real-time kernel implementation in FreeRTOS
5. Performance analysis techniques (10^{th} – 12^{th})
Timing estimation of real-time system
Queue theory applications
Input/output performance
6. Additional application issues (13^{th} -14^{th})
Design for fault tolerance
Software verification & system integration
Performance optimization
7. Case studies: RTOS in practices
FreeRTOS

different course topics in the sequence. The major teaching formats and material employed in this course are presented as the following.

At the beginning, we used a power point slide presentation and class discussion to introduce the definitions of real-time systems. These topics are fundamental for further learning. Thus, it is important to help students to set up a solid and comprehensive foundation. In the class discussion, some questions are designed to enable students to reflect on key concepts in real-time systems, and to encourage active learning. Here are some examples: 1) Are real-time systems synonymous with '*fast or high performance*' systems? 2) "In the statement 'All practical systems are ultimately real-time systems', what is your idea of the *degree of 'real-time'*? 3) "Where could a *response time* requirement of a system come from?". As the discussions proceeded, students gradually deepened their comprehension while, at the same time being inspired to judiciously observe and analyze real-time applications. The homework exercises in this course are extracted from practical application problems. One example question is: considering an automation system for car assembly, describe three different event scenarios, classify the system as hard, firm or soft real-time under each of these scenarios. Such exercises are designed with open-ended questions. The goal is for students to think and give justifications for their answers, and to fortify students' understanding so as to be able to solve problems in practice.

The main purpose of introducing formal requirement engineering techniques is for students to develop an appreciation of automated formal or semi-formal methods in rigorously specifying real-time system requirements. As an example of formal requirement engineering techniques, the requirement document for the "Fuel Management" subsystem from AIRBUS was studied. Students were convinced that the formal specification like State chart was not just a scientific research issue, but in fact had been widely used in industry to avoid the ambiguity caused by conventional text-based specifications. Later on, they showed great enthusiasm in learning formal or semi-formal approaches. In the course evaluation, students made positive comments that the requirement engineering knowledge was one of the most beneficial aspects to them.

More and more microcontrollers (MCUs) are designed to support real-time applications. In this course, the following MCU development boards and Integrated Development Environments (IDEs) are made available to students.

- Atmel SAM4S/4L-EK (ARM Cortex-M4 microcontroller from Atmel)
- STM32 Discovery board (ARM Cortex-M3) from STMicroelectronics
- Keil µVision IDE for ARM programming
- Atmel Studio with Atmel Software Framework (ASF) from Atmel

The last part of this course teaches students how to develop core components for the application. As real-time systems are often applied in 'critical' embedded applications with respect to reliability, safety and security, verification and validation (V&V) is an important issue in real-time application development. An on-going research on the model-based verification and testing has been incorporated into this course so that students could be exposed to the latest V&V advances [4].

RTOSs Survey

Every real-time system contains some operating system (OS) like functionalities: 1) providing an interface to the input/output hardware devices; 2) coordinating virtual concurrency in a uniprocessor environment; 3) enabling true concurrency with multi-core processors and distributed system architectures. In general, a RTOS has three main principal goals: 1) to offer a reliable, predictable, and low-overhead platform for multi-tasking and resource sharing; 2) to make the application software design easier and less hardware bound; 3) to allow engineers with various expertise to concentrate on their core product knowledge. However, a full-featured operating system (OS) is not always the best solution in embedded systems for two main reasons. First, it is hard to meet design constraints of low-end embedded systems, such as system cost and complexity, response time and the punctuality. Second, the underlying MCU resources would be heavily occupied by the system software, with limited resources and computing power left for executing time-critical application programs.

Nowadays, many RTOSs are available from both commercial and academic communities. These RTOSs have different license types, target usage and supported platforms. In order to select the suitable RTOS for our teaching purposes, we first did a survey of existing RTOSs whose target usage is in embedded applications, as shown in Table 2. We follow three selection principles: 1) the license type is free; 2) the RTOS is portable to ARM Cortex series MCUs; 3) the RTOS project is active and well supported. Compared with existing RTOS systems, FreeRTOS is free, richly featured, open source and supports MCUs from multiple manufactures. Moreover, it is easy to access the comprehensive FreeRTOS reference manual and lab exercises.

FreeRTOS Overview

FreeRTOS is chosen to study multitask scheduling policies and real-time interfacing with MCUs. It provides real time scheduling functionality, inter-task communication, timing analysis and synchronization primitives. On top of FreeRTOS, complicated ARM Cortex MCU applications can be efficiently built to meet their hard real-time requirements. It supports organizing these applications as a collection of independent threads of execution.

Since every MCU used in our lab has one processor core, only one thread can be executing at any one time. By computing the scheduling policy designed by the application developer, FreeRTOS is responsible for deciding which task (i.e., one thread of execution) should be executing at a time. Using priority-based scheduling as an example, the application code first assigns higher priorities to tasks that implement hard real-time requirements, and lower priorities to tasks that implement soft or firm real-time requirements. Then, FreeRTOS determines which task should be executing by examining the priority assigned to each task. Thus, hard real-time tasks can always guarantee to be executed ahead of soft real-time tasks.

Table 2: A Survey of Common RTOSs.

Name	License	Platforms	Brief description
FreeRTOS	Free	ARM Cortex, MSP430,STM32	A real-time kernel designed to be small enough run on a MCU, be portable to a large number of MCUs.
QNX	Mixed	ARM, IA32, MIPS, PowerPC, SH-4, xScale	A commercial Unix-like RTOS, targeting general purpose usage, being widely used within a variety of devices including cars and mobile phones.
Window CE or WinCE	Proprietary	ARM, MIPS, x86, SuperH	An OS and kernel developed by Microsoft for embedded systems; a variety of IDE supporting development for WinCE.
Micriumuc/ OS-IIs	Proprietary	ARM, ST32 PIC24	A priority-based pre-preemptive (?) real-time multitasking operating system kernel, featuring unlimited application tasks.
CoDeSys SP Runtime	Proprietary	X86, ARM, Infineon TriCore	Full-featured OS; Component based architecture for customer adaptations to various platforms in industrial automation.
VxWorks	Proprietary	ARM, PowerPC MIPS	Be designed for use in safety and security critical embedded systems requiring real-time, deterministic performance; supports multi-core platform.

Besides ensuring an application meets it's processing deadlines, FreeRTOS can bring other benefits. To help students better understand the usage of this RTOS, two main benefits are introduced through lab exercises. First, the kernel is responsible for timing and provides time-related APIs to the application. This allows the structure of the application code to be simple and the overall code size to be small. Second, the idle tasks which are created automatically by the RTOS can be used to measure spare processing capacity, to perform checks in the background, or to place the processor into a low-power mode for power saving applications.

Use of FreeRTOS in Labs and Projects

Educators have explored hands-on RTOS development for enhancing student learning in real-time systems courses [5], but no one has used FreeRTOS before. The lab assignments based on this open source software are the important experiential component of this course, which aims at giving students a rich hands-on experience in real-time systems design. All course materials are available on line [6].

Three things have been prepared for tutoring each lab session before students start working on lab assignments. First, a list of questions was designed for students to answer during each lab session. These questions served as guidelines to assist students' hands-on learning. Second, the concepts or algorithms related to each lab work, which have been introduced in lectures, were reviewed. Third, as software programing is the main task in each lab, a set of relevant API functions were introduced to students for efficient programming. In the first five weeks, the code templates were also provided to students, with which students only needed to complete certain key statements in C code. A standard demo project which incorporates a MCU development board, simulator and miniature RTOS with all basic features, is provided to students. They used it as the basis to construct application specific projects, which could achieve more efficient development compared with creating the application from scratch. According to the course evaluation report, students gave the feedback that they benefited most from the hands-on programming experience.

Lab assignments on FreeRTOS

The lab assignments using FreeRTOS consist of five key components as shown in Figure 2:

1. Task Management: Eight exercises on creating tasks, changing task priorities, using Block state to create a time delay, combining continuous processing tasks and periodic tasks, defining an idle task hook function, and deleting a task were assigned.

2. Queue Management: Four lab exercises were assigned for students to work on creating a queue for inter-task communication, accessing data from a queue, blocking on a queue, and finding the effect of task priorities on accessing a queue.

3. Interrupt Management: Three lab exercises are assigned for students to get familiar with deferred interrupt processing, the use of binary semaphores / counting semaphores on task synchronization, and the use of queues within an interrupt service routine.

4. Resource Management: Two lab assignments are given on mutual exclusion, mutex, priority inheritance and gatekeeper tasks for protecting resource access.

5. Memory Management: One lab assignment on different memory allocation schemes to illustrate their pros and cons. For example, whether supporting the automatic checking and release of the memory allocated to deleted tasks.

Capstone projects with FreeRTOS

Students have developed multiple projects around FreeRTOS classified into two groups:

1. **Trace analysis**: To better observe the real-time behavior of application tasks, students did multiple projects on instrumenting existing application code with some trace tracking API and executing the instrumented code. The log file records the traces dumped during execution, which is graphically displayed for debugging purposes with a software tool FreeRTOSplusTrace from Percepio. This project helps students gain deeper understanding of scheduling algorithms and conducting performance analysis via graphical traces.

2. **Low power:** FreeRTOS supports not only real-time task management, but also effective low power management. Students did projects on utilizing the idle task hook function to enter the low power mode of the underlying power saving MCU architecture. To calculate the real-time power usage, the supply voltage and the current at different execution modes such as the sleep mode, run mode, deep sleep mode etc. is measured. Two low power MCU boards from ST and Atmel were used in projects. Students have gained the experience of developing low power embedded applications using RTOS.

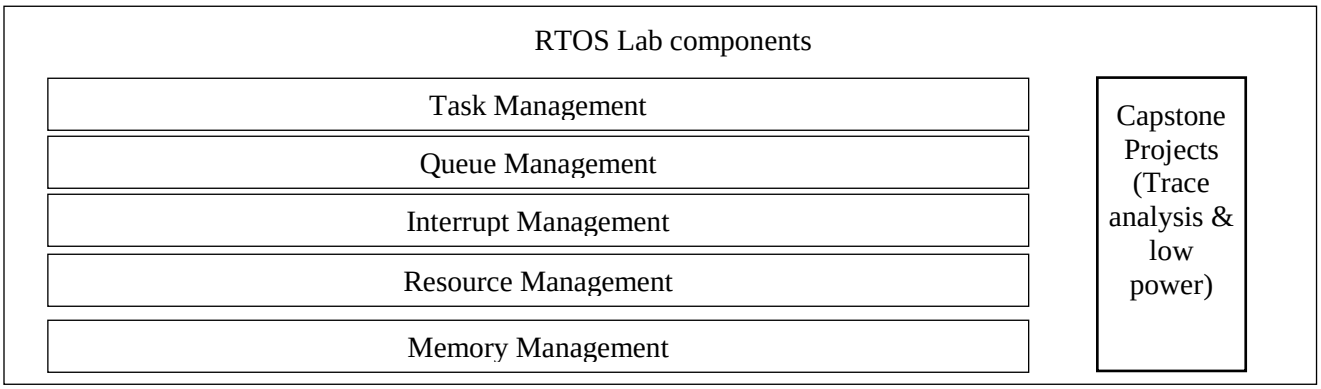

Figure 2: Core Components of Labs.

Conclusion

This paper presents our experiences in teaching the new course "Real-time Embedded Systems Design". The description of this course is first presented. Next, we discuss a survey of common RTOSs and focus on FreeRTOS with which lab assignments and projects are designed. In the future, we will assist students to develop more capstone projects in various real-time embedded applications using FreeRTOS.

References

1. J. Liu, *Real-Time Systems*, Prentice Hall, 2000.

2. P. A. Laplante, S. J. Ovaska, *Real-Time Systems Design and Analysis: Tools for the Practitioner*, 4th edition, Wiley Publisher, 2011.

3. R. Barry, "Using the FreeRTOS Real Time Kernel – A Practical Guide – Cortex-M3 Edition", http://shop.freertos.org/, 2014.

4. N. He, "Incorporating On-going Verification & Validation Research to a Reliable Real-Time Embedded Systems Course," in *Proceedings of 2013 ASEE North Midwest Sectional Conference*, 2013.

5. G.S.A Kumar, R. Mercado, G. Manimaran, and D.T. Rover, "Enhancing student learning with hands-on RTOS development in real-time systems course," in *Proceedings of Frontiers in Education Conference*, 2008.

6. N. He, "Real-time embedded systems design Course".http://mavweb.mnsu.edu/hen/EE498_578_spring14.html, 2014.

Biographical Information

Nannan He is an Assistant Professor in the ECET Department of Minnesota State University at Mankato. She received her Ph.D. in computer engineering from Virginia Tech. Her teaching and research interests are in safety-critical embedded software, real-time systems, and software verification.

Han-Way Huang is a Professor in the ECET Department of Minnesota State University, Mankato. He received his Ph.D. in computer engineering from Iowa State University.