# A COURSE ON ADVANCED REAL-TIME EMBEDDED SYSTEMS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Krista Round

June 2022

COMMITTEE MEMBERSHIP

TITLE: A Course on Advanced Real-Time Embedded Systems

AUTHOR: Krista Round

DATE SUBMITTED: June 2022

COMMITTEE CHAIR: Andrew Danowtiz, Ph.D.
Associate Professor of Computer Engineering

COMMITTEE MEMBER: Joseph Callenes-Sloan, Ph.D.
Assistant Professor of Computer Engineering

COMMITTEE MEMBER: Fred DePiero, Ph.D.
Professor of Computer Engineering

ABSTRACT

A Course on Advanced Real-Time Embedded Systems

Krista Round

This thesis discusses the development of an advanced real-time embedded systems course offered at California Polytechnic State University, San Luis Obispo, which aims to prepare students to design modern complex real-time embedded systems. It describes the goals of the real-time embedded systems curriculum, which includes an introductory and advanced course. Finally, this paper discusses the challenges of creating a successful advanced real-time embedded systems course and proposes changes to the current advanced real-time embedded systems course in response to those challenges.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

Real-time embedded systems (RTES) are used in antilock braking systems (ABS), cruise control, and airbag control systems in cars, pacemakers, insulin pumps, home security systems, and more [1] [2]. Applications for military and medical industries require fault-tolerant and safety-critical design [2]. Due to their adaptability, versatility, and decreasing hardware costs, the market for real-time embedded systems continues to grow. The embedded systems market, including real-time and non-real-time systems, was valued at about 100 billion USD in 2019 and is expected to grow to 160 billion USD by 2026. [3]. This growth is largely driven by increasing use of embedded systems in smart homes, healthcare applications, wearable devices, and the automotive and defense industries [3]. There has also been an increase in publications on designing courses for real-time embedded systems, real-time operating systems, and embedded systems courses in college, potentially indicating an increased interest in offering these courses [4–8].

Often, real-time embedded systems courses focus on the theory of real-time operating systems (RTOS), which leads to a gap in students' theoretical understanding of software topics and practical design and implementation skills [4]. Similarly, projects in real-time embedded system courses are often focused on the theory and not on real-world applications [5]: Students do not get to learn about the challenges of designing the complex systems that are needed for modern commercial systems.

The California Polytechnic State University, San Luis Obispo (Cal Poly) computer engineering department offers two courses on real-time embedded systems. The first is an introductory course called Real-Time Embedded Systems and is offered as

EE/CPE 442. This class includes both important technical information for theoretical understanding of real-time embedded systems and projects that emphasize practical design. The second is an advanced course called Advanced Real-Time Embedded Systems and is offered as EE/CPE 542. EE/CPE 542 currently focuses on an independent project, which is effective in teaching students about designing complex real-time embedded systems, but lacks formal instruction in advanced technical and theoretical information. This thesis discusses the development of a project-focused course on real-time embedded systems proposed for EE/CPE 542 that will enable students to confidently design complex real-time embedded systems.

# CHAPTER 2

# BACKGROUND

## 2.1 What are Real-Time Embedded Systems?

An embedded system is defined as a microprocessor system combined with memory, input/output (I/O) devices, and other mechanical and electrical peripherals [1]. Embedded systems generally perform a specific function as a part of a much larger electrical and mechanical system [1]. For example, an embedded system that controls an airbag system would be responsible for detecting a collision and inflating the airbags. Embedded systems are often small, low-power, low-cost, and single-functioned compared to general-purpose computers [1].

Real-time systems must compute and deliver correct results in a specified period. If a true real-time system fails to deliver the results by the deadline, then the system's output is incorrect even if the results are correct [1]. Using the previous airbag system example, if the car gets into a collision, the airbags must deploy within a certain period to prevent the car occupants from serious injury. If the embedded system fails to meet its hard deadline and the airbags deploy late, it has failed. An airbag is an example of a real-time system and an embedded system.

## 2.2 Real Time Embedded Systems (CPE/EE 442) Course

The Advanced Real-Time Embedded Systems course proposed in this thesis relies on the introductory course, Real Time Embedded Systems as a prerequisite. The first real-time embedded systems course in Cal Poly's two-course sequence is a four unit

(three units of lecture and one unit of lab) course designed to introduce students to modern embedded systems theory, design, and implementation [9]. The goal of the course is to introduce tools and coding techniques for high-performance embedded systems. RTES is offered to fourth-year and Master's level students, so students are expected to be more responsible for their learning. This course focuses on teaching students through labs and projects, with less emphasis on lectures, homework, and exams.

RTES is generally offered Fall Quarter and is a ten-week course. The course meets twice a week, where each class section is a three-hour combined lecture and lab period. There are about 45 minutes of lectures per week, with the rest of the time in a class dedicated to labs, tutorials, and projects. There are no exams or homework for the course.

The course is available to advanced undergraduate electrical engineering, computer engineering, and computer science students and to graduate electrical engineering and computer science students and can be counted as a technical elective for all aforementioned programs. Computer engineering and computer science students are required to take an operating systems class and generally take it before RTES, while most electrical engineering students haven't taken an operating systems course. As a result, computer science and computer engineering students are generally more comfortable with the real-time operating systems topics in this course, which is helpful for understanding real-time embedded systems. Since the course is offered to students with different levels of programming experience, it needs to be flexible enough to be engaging for all. The course offers three different project tracks that students can follow for credit to accommodate these different backgrounds. The first option is the lab track, where students work on predefined labs related to the lecture topic. The labs build off each other, and each lab needs to be completed and working to move

on to the next lab. The second option is the teaching track, which requires students to work ahead of the lab track to figure out how to implement the lab and design a tutorial about the lab and concepts. These tutorials are published as a resource for students on the lab track. Labs and tutorials are completed in small groups of two to thee. The third track is the special project track, where students propose and implement an RTES project over the quarter. Most students follow the lab track, and the rest follow the tutorial track. No students have followed the special project track.

Projects are completed with a Raspberry Pi 3 or 4 with the Raspberry Pi OS. The Raspberry Pi was selected because it can run Linux, its hardware includes multiple CPU cores with single instruction/multiple data (SIMD) support, and it is readily available. Students are expected to purchase their devices, and each project group needs one device. Students also need a virtual machine provided by the instructor for one lab.

The labs and tutorials focus on implementing a Sobel filter for image edge detection on a video. An example of the Sobel filter is shown in Figure 2.1. Students begin the quarter by choosing a video and implementing a grayscale algorithm and a basic Sobel filter in C applied to each frame. The rest of the labs for the quarter introduce students to techniques to improve and measure the performance of the grayscale and Sobel filter using multi-threading, single instruction/multiple data vectors, compiler optimizations, and performance counters. The final project asks students to implement further grayscale and Sobel filter improvements to make it as fast as possible. The schedule for the lectures and labs is shown in Table 2.1.

**Table 2.1: Real Time Embedded Systems Course Overview of Lecture and Lab Schedule.**

| Week | Lecture Topic | Lab Topic | Tutorial Topic |
|---|---|---|---|
| Week 1 | Introduction and RTES background | Linux command line prompts | Makefile with GCC |
| Week 2 | Gray scale and Sobel filter | OpenCV installation and makefile | OpenCV gray scale and Sobel filter implementation |
| Week 3 | Multi-threading | OpenCV gray scale and Sobel filter implementation | Use multi-threading with grayscale and Sobel filter operations |
| Week 4 | CPU inefficiencies and single instruction/multiple data (SIMD) | Use multi-threading with grayscale and Sobel filter operations | Use Arm Neon SIMD engines with grayscale and Sobel filter operations |
| Week 5 | Compiler optimization | Use Arm Neon SIMD engines with grayscale and Sobel filter operations | Use hardware performance counters to measure performance |
| Week 6 | | Week 5 lab | Week 5 tutorial |
| Week 7 | | Use hardware performance counters to measure performance | Final project |
| Week 8 | | Final project | Final project |
| Week 9 | Single instruction/multiple threads (SIMT) | Final project | Final project |
| Week 10 | Final presentation | | |

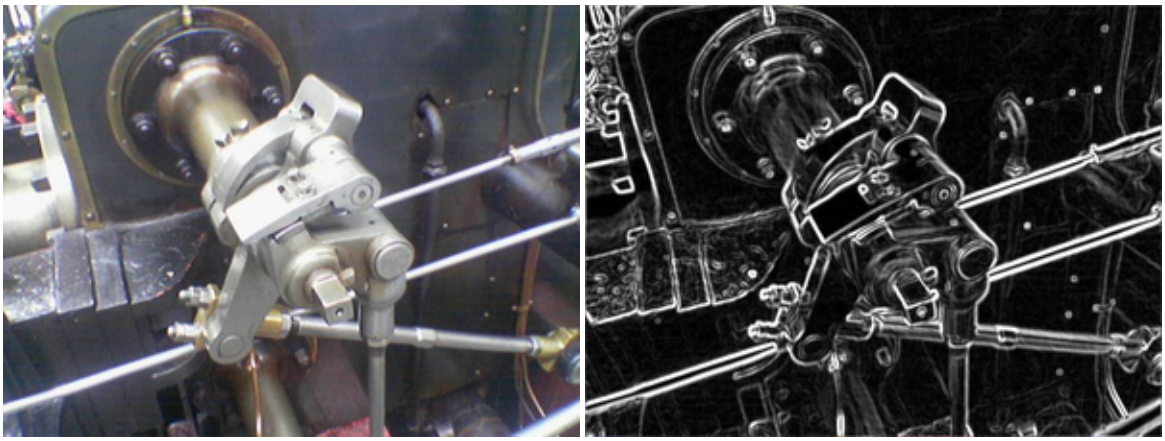


**Figure 2.1: Color Image of Steam Engine and Image With Sobel Filter [10] [11]**

### 2.3 Advanced Real Time Embedded Systems (CPE/EE 542) Course

Once students complete the introductory Real-Time Embedded Systems course, they can choose to continue with the Advanced Real-Time Embedded Systems course. The Advanced Real-Time Embedded Systems course proposed in this thesis is developed based on the current course with modifications discussed in section 4. The Advanced Real Time Embedded Systems course is a four unit (three units of lecture and one unit of lab) course designed to be an advanced independent study of modern embedded systems [9]. The course is almost entirely project-based and focuses heavily on independent research. Advanced RTES is offered to those who completed RTES, and it is generally offered during the Winter Quarter directly after RTES. The goal of the course is for students to use the knowledge from the introductory RTES course and perform an independent project to improve their understanding. This class meets twice a week, where each class section is a three-hour combined lecture and lab period. There are some lectures and one lab at the beginning of the quarter, and the rest of the course is devoted to working on an individual or small group quarter-long project. There are no exams or homework for this course.

The quarter-long project is a student designed, led, and implemented project with ten weeks allocated to completing it. The purpose is to teach students about developing software and low-level drivers to run in a Linux environment and how embedded systems interact with external hardware blocks from a software, circuits, and architecture perspective. This project also exposes students to real-world challenges of using resources on an embedded system and teaches them how to track down and fix these errors. Finally, students must give a presentation and write a report about their project. The project can be done on either a Raspberry Pi 3 or 4 or Zybo Z7 development board. Criteria for the project are that it must use at least one non-processor peripheral on the board, have a software program running on Linux that controls the overall application, have a method to interact meaningfully with the outside world, and be innovative or novel. The schedule for the course is shown in Table 2.2.

## 2.4 Course Material

The selection of an embedded system for EE/CPE 442 and EE/CPE 542 is based on the goals of the real-time embedded systems courses and the labs and projects. For the introductory course, EE/CPE 442, students need an embedded system that they can implement the grayscale and Sobel filter on and implement the techniques to improve and measure the performance of their filter. For the advanced course, EE/CPE 542, a versatile embedded system is needed so students can implement a variety of projects. The cost, accessibility, and available documentation for the embedded system are also crucial for both courses. The embedded systems used in the current version of EE/CPE 542 are also used in the proposed course.

**Table 2.2: Current Advanced Real Time Embedded Systems Course Overview of Lecture and Lab Schedule.**

| Week | Lecture Topic | Lab Topic |
|---|---|---|
| Week 1 | Direct memory access (DMA) | DMA |
| Week 2 | Interface standards | Project proposal |
| Week 3 | Inter-process communication | Project |
| Week 4 | | Project |
| Week 5 | | Midterm project report |
| Week 6 | | Project |
| Week 7 | | Project |
| Week 8 | | Project |
| Week 9 | | Project |
| Week 10 | Final presentation and final report | |

### 2.4.1 Raspberry Pi 3 and 4

The Raspberry Pi 3 and 4 are used for EE/CPE 442 and can be used for EE/CPE 542. It was selected because it has single instruction/multiple data (SIMD) vectors for EE442, and it is readily available. Both versions of the Raspberry Pi have four cores, which are used for the multi-threading lab [12].

The Raspberry Pi 4 has the Broadcom BCM2711 processor compared to the Raspberry Pi 3, which has the Broadcom BCM2837 processor. The Broadcom BCM2711 processor has the ARM A72 core, which is faster. It also has a new GPU, the VideoCore VI, with more features and quicker input/output. The Raspberry Pi 4 has the

option for more RAM and some updated connectors [12]. Although the Raspberry Pi 4 has better performance than the Raspberry Pi 3, the functionality is very similar for this curriculum.

### 2.4.2 Zync Z7 SoC Development Board

The Zync Z7 Development board has the benefit of having both an ARM processor and Field Programmable Gate Array (FPGA), which makes it better suited for independent projects in CPE/EE 542 [13]. The Cortex-A9 processor has two cores [13]. The Zync Z7 is significantly more challenging to setup compared to the Raspberry Pi as much of its configuration and development depends on the Vivado and Vitis software provided Xilinx, which have a steep learning curve. Historically, students who used the Zync Z7 in EE/CPE 542 have spent a significant amount of time trying to get the board set up, which limited the complexity of the projects students were able to complete during the quarter.

## 2.5 Cal Poly Quarters to Semesters Transition

The current structure of the real-time embedded systems curriculum, which consists of the previously discussed EE/CPE 442 and EE/CPE 542 courses, fits Cal Poly's current quarter system. However, the California State University (CSU) Chancellor's Office and Cal Poly President Jeffrey Armstrong announced in Fall 2021 that Cal Poly is transitioning from the quarter system to the semester system beginning in the 2025-2026 school year [14]. The current quarter system consists of three, ten-week quarters in Fall, Winter, and Spring. Some students choose to complete an additional Summer quarter. The future semester system will consist of two 15-week semesters in Fall and Spring and an optional summer term. Starting in the 2022-2023

school year, departments will begin mapping their current curriculum to a semester system curriculum [14]. With the upcoming transition from quarters to semesters, the real-time embedded systems curriculum will eventually need to be re-imagined for the semester system. However, since the transition to the semester system is still in early phases, the proposed Advanced RTES course is designed for the current quarter system. Recommendations for making an RTES and Advanced RTES curriculum for the semester system are discussed in the Future Work section.

## 2.6 ABET Accreditation

The Accreditation Board for Engineering and Technology Inc. (ABET) is an accreditation organization for applied and natural science, computing, engineering, and engineering technology programs at post-secondary schools [15]. Accreditation in The United States is voluntary. ABET accreditation performs a review process periodically to determine if an educational program meets defined standards of quality [15]. Cal Poly's Bachelors of Science program for electrical engineering has been ABET-accredited continuously since 1969, and the Bachelor of Science for computer engineering has been ABET-accredited continuously since 1995. The Masters of Science program for electrical engineering at Cal Poly is not ABET-accredited [16].

As an electrical and computer engineering course, the Advanced Real-Time Embedded Systems course should meet ABET accreditation standards of quality. The proposed changes to EE/CPE 542 should meet the accreditation objectives outlined by ABET, which will contribute to the accreditation of the electrical and computer engineering departments. The accreditation objectives for engineering courses are described in "Criteria for Accrediting Engineering Programs, 2021 – 2022" [17]. Criterion three, "Student Outcomes", defines course objectives that ABET requires programs to meet.

No one course needs to meet all seven objectives, but courses should meet some objectives. The objectives are:

1. "an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics"
2. "an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors"
3. "an ability to communicate effectively with a range of audiences"
4. "an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts"
5. "an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives"
6. "an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions"
7. "an ability to acquire and apply new knowledge as needed, using appropriate learning strategies"

This course is designed to fulfill objectives one, four, and five.

Additionally, the Cal Poly electrical and computer engineering departments define program education objectives that must be met for ABET accreditation [17]. The electrical engineering objectives are to prepare graduates to:

1. “excel in the electrical engineering profession”
2. “embrace life-long learning as a necessary component to remain current in their profession”
3. “pursue graduate degrees for enhanced skills and opportunities”

The computer engineering objectives are to:

1. “make positive contributions to society and the practice of computer engineering by applying foundational knowledge and the engineering process to solve engineering problems”
2. “work in an individual or team environment in a socially responsible manner”
3. “engage in lifelong learning through continued professional development or graduate studies”
4. “communicate effectively and demonstrate leadership”

This course is designed to fulfill the second and third electrical engineering program education objectives and the third and fourth computer engineering program education objectives. The specifics of how this course meets ABET’s and Cal Poly’s objectives are in section 4.2.

# CHAPTER 3

# LITERATURE REVIEW

## 3.1 An Interdisciplinary Curriculum on Real-Time Embedded Systems

An example of a real-time embedded systems curriculum is discussed in "An Interdisciplinary Curriculum on Real-Time Embedded Systems," created at Kansas State University [4]. The design and implementation of embedded systems requires knowledge of computer science, electrical engineering, computer engineering, mechanical engineering and more. Traditional embedded systems courses emphasize hardware construction over software design [4]. The curriculum at Kansas State University applies an interdisciplinary approach to teach students about hardware and software integration. This curriculum aims to provide students with broad knowledge and interdisciplinary skills to build complex embedded systems. This curriculum also seeks to close the gap between students' conceptual understanding of theory and practical implementation by providing students with laboratory assignments similar to actual implementations in the engineering industry.

This curriculum consists of four-semester long (15 weeks) courses intended to be taught to advanced undergraduate students and early graduate students from computer science and various engineering majors. The first course provides background knowledge about real-time electronics and programming. The second course focuses on implementing a simple real-time system to build a Controller Area Network (CAN), which is used in industrial automation, automotive electronics, agriculture, and marine craft. The third course teaches traditional real-time embedded systems techniques such as scheduling theory and project phases, including requirements, design,

implementation, and verification techniques. The fourth course is an interdisciplinary team project where students design and implement a complete embedded system. Students' evaluation surveys showed the course performed well for course process, student learning, and overall [4].

### 3.2 A Course on Advanced SOC FPGA in Embedded Systems

An example of an advanced embedded systems curriculum is discussed in "A Course on Advanced SOC FPGA in Embedded Systems" and was created at Oakland University [7]. This curriculum was designed to meet the industry demand for engineers who can build complex embedded systems and use advanced field-programmable gate arrays (FPGAs) for embedded systems. FPGAs are helpful for embedded systems projects because they provide flexibility, faster development cycles, and high performance. This curriculum aims to teach students how to program in hardware description language (VHDL) and C and how to design using an FPGA and microcontroller core.

The curriculum consists of one semester-long course taught to senior undergraduate computer engineering students and graduate students. Students complete small lab projects and give a 15-minute presentation to the class about advanced topics in embedded systems. These presentations expose students to the most advanced technology and applications. The students also benefit from learning how to perform a literature search and teach themselves advanced topics.

The authors discuss some of the challenges of teaching an advanced embedded systems course. One significant challenge is that textbooks do not cover the latest software tools, standards, requirements, and hardware versions. Some system on a chip (SoC) boards get revised annually, so information about the board provided by a textbook

may not be accurate a year after publication. As a result, some labs provided by embedded systems textbooks may not work on new boards or with the latest version of a vendor's board-specific development software. The authors solve these issues by making changes to the course annually. Before starting the course, the authors correct software issues for labs and create notes on updated standards and requirements.

### 3.3 Conclusions from Literature Review

In the Kansas State University real-time embedded systems curriculum, the third course teaches advanced real-time embedded systems topics, while the fourth course is a team project. The authors report that students learned a lot from the first, second, and third courses focused on theory and projects and did not report student learning from the fourth course. As a result, our proposed Advanced Real-Time Embedded Systems curriculum will include theoretical topics in the form of lectures and a long student-designed project.

The paper from Oakland University provides a lot of insight into how to address changes and advancements in embedded systems technology in a course. Students are required to give a research presentation to the class about a current advanced topic in embedded systems. Since the material students present changes annually, the presentations keep the course updated with advanced technology. The authors reported that the students were highly interested in these presentations and learned more being taught by other students. Some students also benefited from the research presentations because they helped guide them in finding a topic for their graduate research. To keep the proposed course at Cal Poly up-to-date with current RTES research, it will also include a research presentation about a current topic. This

research presentation assignment will ensure that the material in the proposed course is relevant without the faculty having to redesign lectures annually.

The paper from Oakland University also discusses the importance of using an FPGA for RTES research. Using an FPGA for the student-designed project allows the projects to be flexible, have fast development cycles, and have high-performance. Students who use a development board with an FPGA and a CPU, such as the Zybo Z7, will also learn how to program in C and design using an FPGA and microcontroller core. As a result, the Zybo Z7 board will continue to be encouraged for students to use on their project in EE/CPE 542.

# CHAPTER 4

# CURRICULUM

## 4.1 Goals for Advanced Real Time Embedded Systems Course

The goal of the proposed Advanced Real Time Embedded System course is to allow students to apply their background knowledge of embedded systems theory taught in this curriculum in a large student-designed project. This project will focus on teaching students advanced custom hardware/software interaction for high-performance embedded systems.

This course is intended to be student lead. Although the course will include some structured material, the goal is for students to be able to focus on their own advanced embedded systems interests. Another goal of this course is to help students learn how to overcome the real-world technical challenges, errors, and frustrations that come with designing a complex system. Many of these challenges and problems are new and require new methods for solving them. Students will be responsible for designing and implementing their projects and finding the resources they need to fix any errors.

Finally, this course needs to meet the requirements and objectives set by the electrical and computer engineering department and needs to meet some of the objectives required for ABET accreditation. According to the Cal Poly electrical engineering and computer engineering catalog, the course is an "[a]dvanced study and application of modern embedded systems" and includes topics such as "[m]emory bandwidth matching, clock-domain crossing, IP creation and verification, and student-led lectures on modern System on Chip (SoC) design topics." Finally, the course should include "[b]uilding a prototype embedded system" [9].

## 4.2 Challenges of Creating and Teaching Advanced Real Time Embedded Systems

As commercial embedded systems become more complex and advanced, there is a demand from universities and educators to design courses that include realistic and complex projects. Meeting these expectations is challenging because students have a variety of technical backgrounds, and courses have limited resources and time. These challenges encourage courses to be designed with as much material, labs, projects, and homework as possible to maximize the content covered in a course. However, increasing the time requirements and expectations for the course also increases students' stress levels [8]. As a result, this course's first and most significant challenge is to provide advanced lecture material and complex and realistic projects while preventing student overload.

This course is open to students from various majors and for undergraduate and graduate students. The challenge to this course is designing it so that the material and structure are engaging and challenging for students who have a wide variety of technical backgrounds and different levels of research and independent project experience.

## 4.3 Approach for Course Structure and Material

To teach students about advanced real-time embedded systems topics, this class will include weekly lectures on theoretical topics. This material follows the material from EE/CPE 442 Real Time Embedded Systems but includes more advanced topics needed to understand the basic of operating systems and how to use peripheral devices. Lecture material is delivered with slides.

In its current incarnation, the Advanced RTES course includes a quarter-long independent project where students design and implement a complex embedded system. This quarter-long project gives students enough time to design and implement a complex real-time embedded system of their choosing. Since the project is student-designed, it is adaptable to an individual student's or group's skills and can be as ambitious as they want. The hope is that this project will encourage students to try ambitious projects that aren't guaranteed to succeed in a low-pressure class where they can focus on the learning instead of the project outcome.

Finally, this class will require students to give a short presentation to the class on a current embedded systems topic or technology. This assignment keeps the course up to date with recent research and technologies without requiring a course redesign every year.

By adding lecture material and research presentations in addition to the quarter-long project in the current version of EE/CPE 542, students will have more technical knowledge that they can use in their project and the engineering industry. The Advanced Real Time Embedded Systems schedule is shown in Table 4.3. The grading rubric is shown in Table 4.4. The details of the topics, requirements, assignments, and deadlines for all aspects of this course will be discussed in more detail next, and the Syllabus for the course is in Appendix A.

**Table 4.3: Proposed Advanced Real Time Embedded Systems Course Overview of Lecture and Lab Schedule.**

| Week | Tuesday | Thursday |
|---|---|---|
| Week 1 | Introduction | Develop project proposal, approve research topic, and sign up for presentation date |
| Week 2 | RTOS background | Project proposal due |
| Week 3 | Task scheduling | Student presentations |
| Week 4 | Task scheduling | Student presentations |
| Week 5 | Resource management | Student presentations |
| Week 6 | Resource management | Student presentations and midterm report due |
| Week 7 | Inter-process communication | Student presentations |
| Week 8 | Buses | Student presentations |
| Week 9 | Board Input/Output | Student presentations |
| Week 10 | Final presentation, final report, and journal due | Final presentation, final report, and journal due |

**Table 4.4: Proposed Advanced Real Time Embedded Systems Course Grading.**

| Category | Weight (Percent) |
|---|---|
| Research presentation | 10 |
| Final project proposal | 5 |
| Final project midterm report | 5 |
| Final project high level application | 20 |
| Final project low level application | 20 |
| Final project demo and presentation | 10 |
| Final project journal | 5 |
| Final project report | 15 |
| Final project difficulty | 10 |

## 4.4 Quarter-Long Project

The quarter-long project is the most significant aspect of the course. Students will have ten weeks to develop an advanced real-time embedded systems project and present it to the class at the end of the quarter. The project is intended to be open-ended to encourage students to focus on their interests and be ambitious. The project can be completed individually or in small groups of up to four. However, working in small groups is highly encouraged to improve teamwork skills. In addition, students who work on an independent project are encouraged to work on the project in class. Working on the project in class is encouraged to facilitate collaboration with other students who are working on a similar project with overlapping software and hardware

and to help with troubleshooting. In previous offerings of the course, most of students chose to work in a group. The Quarter-Long Project Description is in Appendix B.

### 4.4.1 Project Requirements

The project must use one non-processor peripheral on the board or use external hardware components connected over a serial bus. Using a non-processor peripheral is essential for thoroughly learning about the complexities of embedded systems. The project must also have the means to interact with the outside world, through a display or otherwise.

The most important requirement for the project is that it is innovative and ambitious. This project aims to provide students with the opportunity to invest a long time in a complex project. There is less stress and pressure on the performance or success of the project than there would be in the engineering industry, on a senior project, or on a graduate thesis. As a result, students are encouraged to be ambitious even if they are unsure that the project can be completed within the quarter. Goals for the project can be renegotiated throughout the quarter.

### 4.4.2 Project Deliverables

Students begin formulating their projects on the first day of class when the project is introduced. They have until the end of the second week of the course to develop their project proposal and identify their project group. In the report, they must include a one or more page proposal outlining the project's significance or purpose, the system they are planning to build, what hardware they think they will need, what software modules they will construct, and objective goals for their project. As previously mentioned, these goals can be renegotiated during the course. Still, the

project proposal is intended to make sure students have thoroughly thought through the project before they begin implementation. They must also discuss the accessibility of the project, including who will use the project and how they can design the project to ensure it reaches a broad audience.

Students will also keep a journal of the work they are completing for the project. For days that the students work on their project, they will record the approximate time they spent on it, along with the work completed and any issues encountered. Screenshots, pictures, website links, and diagrams showing their work, resources, and issues are encouraged. This journal is informal and separate from their final report. The purpose is for students to track their work to understand what aspects of the project they're spending the most time on. For group projects, the journal will also help measure the effort of each group member. Finally, this journal will be helpful at the end of the quarter when composing a final presentation and project. This journal will be submitted at the end of the quarter.

At the end of week 6, students will submit a midterm report. This will include an updated list of the hardware and software they are using. Students will also write about their progress, including any technical challenges they have had to overcome and any changes in their project scope and goals for the end of the quarter.

At the end of the quarter, students will submit a final report. Students can submit a formal final report, which includes information about their design process and decisions, bill of materials, system architecture diagrams, measurable outcomes, etc. Students also can create an instruction report for their final report. The instruction report will teach others how to do the final project. Beyond including step-by-step instructions for implementing the project, it must include context about why these steps are necessary and what common issues others may encounter while following the instruction report. Both the formal final report and instruction final report should

show awareness of why certain design decisions were made and why certain implementations did or didn't work and include a discussion of the accessibility of the project.

Finally, students will give a 20-minute final presentation during the last week of class, outlining what their project accomplishes, what components they used, what challenges they faced, and what resources they used. If students have a working project, they should demo it to the class. At the end of the quarter, students will have submitted a project proposal, midterm project report, final report, journal, and presentation, all of which will be considered for the project's final grade.

### 4.5 Research Topic Presentation

For the research topic presentation, students will choose a current topic in real-time embedded systems that they are interested in and that is different from their quarter project. Choosing a different topic from their project is intended to increase the diversity of material covered in the course and avoid repetition. The presentation can be completed individually or in small groups. The instructor must approve their topic at the beginning of the quarter. Presentations will occur on Thursdays at the beginning of class.

Students will then develop a 30-minute presentation about their topic. The presentation must include background about the topic, including why it is significant and what the current challenges and limitations are. They must also discuss recent advancements or changes related to the topic and any proposed future work. They are encouraged to use various resources, but at least two of them must be recent technical published papers. After the presentation, there will be time for students to ask

questions related to the presentation. The Research Topic Presentation Description is in Appendix C.

## 4.6 Course Topics

### 4.6.1 Textbooks

The current curriculum for EE/CPE 442 and EE/CPE 542 doesn't use a textbook for the course material. Providing students with a textbook is helpful for students interested in learning more about the lecture material, and it is also a single consolidated source for reliable information. To limit the cost of the course for students and improve accessibility, all selected textbooks are free through the Cal Poly library. Three textbooks were used for lecture material information.

The first textbook is "Real-Time Embedded Systems" by Jiacun Wang published in 2017 [1]. This textbook includes introductory and advanced real-time embedded systems topics, such as embedded systems hardware components, operating systems, task scheduling, resource sharing and access control, practical issues, and more. Much of the lecture material references this textbook.

The second textbook is "Embedded Systems Architecture" by Tammy Noergaard published in 2013 [18]. This textbook includes information about standard embedded system board hardware such as board memory, board I/O, and board buses.

The third textbook is "Multicore DSP" by Naim Dahnoun published in 2018 [19]. This textbook was specifically created for projects with the TMS320C66x system on a chip (SoC), but it does include some general information about inter-process communication that is relevant to real-time embedded systems.

### 4.6.2 Lecture Material

Lecture material for the entire course, which is in the form of slides, is included in Appendices D through L. The first week of class will not have any technical course material. On the first day of class, students will review the syllabus, required course materials, quarter-project, and research topic presentation. Students will be provided with the requirements, assignments, and deadlines for the project and presentation. The remaining time on the first day and the second day of class will be for students to join groups, begin formulating project ideas, work on their project proposal, begin formulating their project idea, approve their project presentation with the instructor, and sign up for a presentation date. By the end of week 1, students should have a presentation idea approved and a presentation date selected.

The second week of class will include real-time operating system topics. The topics will be mostly review for computer engineering and computer science students who have taken an operating systems class. This material will be new for electrical engineers who haven't take an operating systems class. This lecture was included in the course to help students unfamiliar with operating system components and terminology. In addition, this lecture is important for those familiar with general-purpose operating systems to understand the important differences between a general-purpose and real-time operating system. Other important material in the lecture is process and memory management, interrupts, clocks and timers, and examples of common RTOSs.

The third and fourth weeks of class will discuss task scheduling. The scheduler is an important component of all operating systems, so understanding its behavior is crucial to understanding how to use a real-time embedded system. Since this topic is so significant, two weeks will be allocated for it. This lecture focuses on

different scheduling algorithms. The two types of scheduling algorithms discussed are uni-processor scheduling, which is scheduling tasks on a single processor, and multi-processor scheduling, which is assigning tasks to a specific processor. The lecture includes clock-driven scheduling, which is scheduling based on the time a task needs to be completed, and priority-driven scheduling, which is scheduling based on a priority the operating system assigns to each task. Examples of the algorithms are also covered.

The fifth and sixth week of class will discuss resource management. In an embedded system where resources such as memory and I/O units must be shared, resource management can affect how quickly tasks are executed and can prevent tasks from executing in time. As a result, understanding resource management is important for making high-performance embedded systems. Two weeks are spent on this material. The lecture covers different resource management protocols and shows how poor resource management can cause important tasks to miss their deadlines. Examples of the protocols are also covered.

The seventh week of class will discuss inter-process communication. Inter-process communication is necessary for exchanging information between different threads, processes, and processors. Efficient inter-process communication increases bandwidth and reduces latency, which is important for high-speed applications. The lecture covers inter-process communication methods such as semaphores, shared files, shared memory, message queues, pipes, and sockets. The lecture also covers the benefits and drawbacks of different inter-process communication methods and different scenarios where certain types of inter-process communication work better.

The topics from weeks two through seven are mainly operating systems topics. The remainder of the course will focus on embedded systems hardware topics necessary for communication with the outside world.

The eighth week of class will discuss embedded system buses. The lecture covers different types of buses, what types of signals they carry, and their hardware structure. The focus of the lecture is bus arbitration schemes, which are used when multiple devices are connected to a bus to determine which device can control the bus. Finally, the lecture covers bus performance and methods to improve bus performance.

The ninth week is the last week of instruction before final presentations during week ten. The ninth week of class will discuss board I/O. Understanding board I/O is necessary for connecting an embedded system with external devices. The quarter project requires students to interact with the world in a meaningful way, so board I/O will be important for many students who will likely use external hardware. The lecture covers standard I/O hardware, serial, parallel, asynchronous, and synchronous I/O, and I/O performance.

## 4.7 ABET Criteria

As discussed in the Background ABET Accreditation section, the electrical and computer engineering departments at Cal Poly are ABET accredited and therefore, courses in the electrical and computer engineering departments must meet certain criteria [16]. These criteria are outlined by ABET in "Criteria for Accrediting Engineering Programs, 2021-2022" and by the Cal Poly electrical and computer engineering departments through their program education objectives. This section discusses how the proposed course previously outlined meets these objectives and how these outcomes are documented.

### 4.7.1 Criteria for Accrediting Engineering Programs

The first criteria is "an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics" [17]. Through the quarter-long project, students must design their system, create a system diagram, and describe the design decisions they made. They must then implement the design and show how they worked through any technical challenges. All of the daily work is documented in the journal, and the entire system is included in a formal report.

The second criteria is "an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives" [17]. Although the quarter-long project and research topic presentation aren't required to be completed in groups, students are highly encouraged to do so, and most students chose to work in groups for the project in past offerings of the course. Students who work in a group for the project will need to work as a group to decide on their project and set objective goals for their project, which will be included in their project proposal. Throughout the quarter, if a group finds they cannot meet their initial goals, they have the option to decide together to renegotiate their goals and submit them again during the midterm report and final report. They must also demonstrate during their final report and demo how their project meets their objectives. The journal that students also submit shows the work done by group members, outlining how each member contributed to the group's overall work. Students who choose to work on the project independently are still encouraged to collaborate with other people in the class who are working on a similar project or with similar software or hardware. The goal of this is to create an environment where students collaborate on ideas and solve problems together in

groups larger than their independent project group. The final project, journals, and final presentations can be submitted to ABET to meet this objective.

The third criteria is "an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts" [17]. As a part of the project design process, students need to identify who will use the project and how they are going to ensure it can reach a broad audience. This requires students to assess the use and the societal impact of their project. The goal is to encourage students to understand the impact their design choices have and to take responsibility for making those design choices ethical. The accessibility analysis is included in the project proposal and final report and can be submitted to ABET to meet this objective.

### 4.7.2 Program Education Objectives

The first objective of the electrical engineering program education objectives is "embrace life-long learning as a necessary component to remain current in their profession" [20]. Similarly, the first objective of the computer engineering program education objectives is "engage in lifelong learning through continued professional development or graduate studies" [21]. Two of the essential goals of this course are to allow students to perform independent work on a real-time embedded systems topic that interests them and to perform research on an advanced topic. Students will learn how to identify and formulate a technical project, design and implement it, and write a report about it. Through this process, they will hopefully improve their confidence that they can complete ambitious and independent projects and be encouraged to continue their research interests with graduate education. Even if graduate educa-

tion is not in their current interests, they will hopefully be encouraged to continue to pursue their academic interests independently.

The second objective of the computer engineering program education objectives is for students to "communicate effectively and demonstrate leadership" [21]. Students will complete two separate presentations during this course, first on their research topic and then their project. The research topic presentation aims to improve students' skills by performing a literature search and presenting the current research along with other sources. The project presentation seeks to improve students' skills at presenting their own designs and results clearly and concisely.

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

This thesis proposes changes to the current Advanced Real Time Embedded Systems course, EE/CPE 542. The proposed changes capture elements from similar courses discussed in the Literature Review, section 3.3. Based on the paper from the Kansas State University course, which showed the benefit of having both theoretical and project-based learning, our proposed Advanced Real-Time Embedded Systems curriculum will include theoretical topics in the form of lectures and a long student-designed project. The paper from Oakland University provides a lot of insight into how to address changes and advancements in embedded systems technology in a course by including a research topic presentation, which is also included in our proposed course. The paper from Oakland University also discusses the importance of using an FPGA for RTES research. As a result, the Zybo Z7 board will continue to be encouraged for students to use on their project in EE/CPE 542. These modifications also provide students with more resources, such as lectures and several textbooks. The hope is that these changes will maintain the independent and student-focused aspects of the course while adding more material that's helpful for theoretical understanding and expands students' knowledge of current research. Hopefully, this proposed course will enable students to confidently design complex real-time embedded systems.

## 5.2 Future Work

Given the upcoming transition from the quarter system to the semester system, the real-time embedded system curriculum, consisting of EE/CPE 442 and EE/CPE 542 will need to be adjusted. However, departments will not begin mapping the current curriculum to the adjusted semester curriculum until the 2022-2023 school year. Hopefully, the electrical and computer engineering department will choose to keep the real-time embedded systems curriculum to offer to students as an advanced undergraduate and graduate elective. To change to the semester system EE/CPE 442 and EE/CPE 542 could be combined into a single 15-week course. Although this would reduce the total amount of time spent on the material, there would still be enough time to cover all necessary topics. The first seven weeks of the course could cover lectures and labs from EE/CPE 442 up until lab/tutorial 5, which covers performance counters. The remainder of the quarter, eight weeks, could be spent on what is proposed as the quarter-long project in EE/CPE 542. To ensure that as much time as possible is spent on implementing the quarter-long project, students could begin formulating it in the first seven weeks of the semester. The research topic presentations could be spread throughout the entire semester. This structure would maintain the goals and material from the current courses.

# REFERENCES


[1] Jiacun Wang. *Real-Time Embedded Systems*. First Edition. John Wiley Sons, Ltd, 2017. ISBN: 9781119420712. DOI: https://doi.org/10.1002/9781119420712.

[2] Jane Liu. *Real-Time Systems*. First Edition. Prentice Hall, 2000. ISBN: 0130996513.

[3] *Industry Trends*. 2020. URL: https://www.gminsights.com/industry-analysis/embedded-system-market.

[4] Mitchell Neilsen. "An Interdisciplinary Curriculum On Real Time Embedded Systems". In: *2002 Annual Conference*. 10.18260/1-2–10046. https://peer.asee.org/10046. Montreal, Canada: ASEE Conferences, 2002.

[5] Nannan He and Han-Way Huang. "Use of FreeRTOS in Teaching Real-time Embedded Systems Design Course". In: *2014 ASEE Annual Conference & Exposition*. 10.18260/1-2–23240. https://peer.asee.org/23240. Indianapolis, Indiana: ASEE Conferences, 2014.

[6] Henry Chaya. "An Embedded Systems Course Using The Oopic Microcontroller". In: *2002 Annual Conference*. 10.18260/1-2–10898. https://peer.asee.org/10898. Montreal, Canada: ASEE Conferences, 2002.

[7] Subramaniam Ganesan and Fayadh Alenezi. "A course on Advanced SOC FPGA in Embedded systems". In: *2022 ASEE - North Central Section Conference*. https://peer.asee.org/39224. Pittsburgh, Pennsylvania: ASEE Conferences, 2022.

[8] J.W. Bruce and Ryan A. Taylor. "Using Information Gap Learning Techniques in Embedded Systems Design Education". In: *2017 ASEE Annual Conference & Exposition*. 10.18260/1-2–29078. https://peer.asee.org/29078. Columbus, Ohio: ASEE Conferences, 2017.

[9] *2021-2022 Catalog Electrical Engineering (EE)*. URL: https://catalog.calpoly.edu/coursesaz/ee/.

[10] *A color picture of a steam engine*. 2008. URL: https://en.wikipedia.org/wiki/Sobel_operator.

[11] *The Sobel operator applied to that image*. 2008. URL: https://en.wikipedia.org/wiki/Sobel_operator.

[12] *Raspberry Pi Documentation: Processors*. URL: https://www.raspberrypi.com/documentation/computers/processors.html.

[13] *Zybo Z7 Reference Manual*. URL: https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual?redirect=1.

[14] Ashton McIntyre. *Cal Poly, last CSU operating on a quarter system, announces transition into semester system*. 2021. URL: https://www.ksby.com/news/local-news/cal-poly-last-csu-operating-on-a-quarter-system-announces-transition-into-semester-system.

[15] *What is Accreditation?* URL: https://www.abet.org/accreditation/what-is-accreditation/.

[16] *Accredited Programs*. URL: https://amspub.abet.org/aps/name-search?searchType=institution.

[17] *Criteria for Accrediting Engineering Programs, 2021 – 2022*. URL: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2021-2022/.

[18] Tammy Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Second Edition. Newnes, 2013. ISBN: 978-0-12-382196-6. DOI: https://doi.org/10.1016/B978-0-12-382196-6.00016-9.

[19] Naim Dahnoun. *Multicore DSP: From Algorithms to Real-Time Implementation on the TMS320C66x SoC*. First Edition. Wiley Publishing, 2018. ISBN: 1119003822.

[20] *Cal Poly Electrical Engineering ABET Accreditation*. URL: https://ee.calpoly.edu/academics/abet.

[21] *About Us*. URL: https://cpe.calpoly.edu/about/.

[22] *Real-time Embedded System Design Graduate Certificate*. URL: https://catalog.k-state.edu/preview_program.php?catoid=2&poid=232.

[23] *Automotive application guide: Driving the future of automotive electronics*. URL: https://www.infineon.com/dgdl/Automotive_Application_Guide_2016_BR.PDF?fileId=5546d461584d1a55015887056dff07ea.

[24] *ASEE Peer*. 2022. URL: https://peer.asee.org.

# APPENDICES

## Appendix A

## SYLLABUS

**EE/CPE 542 Advanced Real Time Embedded Systems**

**Course Description:** Advanced study and application of modern embedded systems. Memory bandwidth matching, clock-domain crossing, IP creation and verification, and student-led lectures on modern System on Chip (SoC) design topics. Building a prototype embedded system.

**Prerequisites:** EE/CPE 442

**Textbook:** Real-Time Embedded Systems by Jiacun Wang, 2017

**Lecture Time:**

**Instructor:**

**Office Hours:**

**Learning by Participating**
Courses are most successful when there is an active dialogue between the professor, your classmates, and you. If you ever have a question in class, ask! If you have any non-private questions outside of class and office hours, post to Piazza! If you think you know the answer to another student's question, don't be afraid to respond! We want to foster a supportive and collaborative learning environment, so disruptive in-class behavior or any personal attacks on another student will result in you being asked to leave. I do not answer course-related questions over email.

**Collaboration**
Collaboration and knowing when to ask for help are key parts of being a successful engineer. I encourage you to discuss labs with your classmates. Your team may ask high-level questions or debugging questions to other teams for labs. Sharing HDL, entire C functions, or write-up text is not permitted. You also may not copy HDL or C from the internet unless explicitly permitted in the assignment.

**Grading:**

| Category | Weight |
|---|---|
| Research presentation | 10% |
| Final project proposal | 5% |
| Final project midterm report | 5% |
| Final project high-level application | 20% |
| Final project low-level application | 20% |

| | |
|---|---|
| Final project demo and presentation | 10% |
| Final project journal | 5% |
| Final project report | 15% |
| Final project difficulty | 10% |

**Course Topics**

- RTOS kernels, interrupts, clocks, timers, memory management, and process management
- Task scheduling
- Resource management
- Inter-process communication
- Buses
- Board I/O

**Course Supplies**

- Raspberry Pi 3 or 4 or Zybo Z7 development board
- Access to a Linux computing environment (VM, dual-boot, the Pi itself)

**Success with Integrity**

In accordance with Cal Poly's Standards for Student Conduct, any determination of cheating, plagiarism, or other academic dishonesty will at a minimum result in an "F" grade on the assignment and a report to the OSRR.

**ADA Accommodations**

If you require special accommodations while taking this class, please discuss your needs with me as soon as possible. Persons who wish to request disability-related accommodations should also contact the Disability Resource Center in Building 124, Room 119. Phone: (805) 756-1395 or (805) 756-6266 (TTY). Office hours are Monday-Friday from 8:00 AM – 4:30 PM. Some accommodations may take up to several weeks to arrange.

**Diversity Statement**

To build products, devices, and services that meet the needs of all stakeholders, it is imperative that engineering design processes be influenced by a wide range of perspectives from people of different cultural, religious, ethnic, gender, and racial identities and physical and learning abilities. All of these perspectives must be treated with equal respect and consideration. To foster a learning environment where this type of collaboration is possible, all students in this class are required to treat each other with respect.

Further, to combat "designed for (people like) me" syndrome, all project final reports/instructables should include a brief section on Accessibility, describing steps made to ensure that your project would be usable by a wide range of individuals, and describing how the design could be modified to be accessible to an even wider range of users.

**Fine Print**

These course policies are designed to create a fun and effective learning environment. With that goal in mind, I reserve the right to modify course policies in a reasonable manner to improve the course. If any aspect of the course is not working for you, please send me feedback!

# Appendix B

## QUARTER-LONG PROJECT DESCRIPTION

**Skills**
This assignment aims to help you practice the following skills that are important in understanding how software and hardware work together in a computing system.

- Developing software to run in a Linux environment
- Understanding how a modern processor interacts with external hardware blocks from a software, circuits, and architecture perspective
- Experiencing the real-world challenges of efficiently using resources on an SoC
- Tracking down and fixing errors in these systems
- Succinctly and completely communicating technical results

**Knowledge**
This assignment will also help you recognize why it's important to follow best practices in programming while giving you invaluable hands-on experience in working with low-level software on complex SoC hardware. You will be able to use this knowledge to create novel systems, understand the design challenges of using non-processor blocks in a system, and become a valuable member of a computer-system design team.

**Task**
Your task is to propose, implement, document, and demonstrate an application of your choice on your Raspberry Pi or Zybo board. The project can be completed individually or in small groups with a maximum of four people.

**Criteria for Success**
Successful assignments will have the following characteristics:

- Use at least one non-processor peripheral connected over a serial bus on your Raspberry Pi or Zybo (DMA engine, graphics processor, SPI interface, etc.)
- A means for your application to display results/transfer data/or otherwise interact meaningfully with the outside world
- An innovative, novel, or otherwise comprehensive system worthy of being a quarter-long project
    - Be ambitious! We can renegotiate the project scope throughout the class.
    - Anyone who gets a peer-reviewed publication or successful Kickstarter from their project gets an A in the class!

**Project Milestones**
- Thursday of week 2: A 1-page written project proposal outlining what system you're planning to build, what VHDL module you will construct, the project goals, etc. Please also consider and discuss the accessibility of the project. Who is going to use the project? How can you ensure it reaches as broad of an audience as possible?
- Thursday of week 6: A midterm report and Bill of Materials
- Weekly in-class meetings.
- Last Day of Class

- Final project presentations: During the last day of class, you'll have the opportunity to present your final project. The last day of class will be dedicated to a final project showcase, where you'll get to show off your work.
- Final report: The final report is due on the last day of class. For the final report, you will be writing an "instructable" or tutorial to teach others how you built your system. Your final report must also include a section on "accessibility," which describes steps you took to ensure your project could be used by as broad an audience as possible. This section should also include any potential modifications that could be made to your design to make it more accessible. You are highly encouraged to put this tutorial online at a place like instructables.com.
- Journal: The journal is informal documentation of each individual's work on the project, and you should complete it throughout the quarter.

**Final Project Presentation**

Create a presentation highlighting your quarter-long project. It may be up to 10 minutes in length (although it doesn't need to be that long) and should explain:

1) Big Picture: What did you (try to) build and why? What is the significance of this project?

2) System architecture: What parts of the hardware do you use? For Pi: do you access any of the on-chip accelerators? Do you use external components? For ZYNQ: did you add any custom hardware?

3) Challenges: What challenges did you face, and (how) did you solve them?

4) System demo

**Final Project Report**

For this report, you have two options:

1) Create an Instructable

2) A traditional final report/design document

If you choose the Instructable option, your job will be to write a document that teaches others how to make your project. Remember that an Instructable is more than just a simple recipe for making your project; it shouldn't be limited to things like "Step 1:Install OpenCV using 'sudo apt install opencv-dev,' Step 2: Download this included code, Step 3: Run it, Step 4: You're done." Make sure to provide plenty of context for why readers should take different steps. Why do they need OpenCV? What does your included code do? Etc. If you'd like, you can publish your instructable at instructables.com and just provide me a link or printout here.

Please dispense with cover pages if you choose to do a more formal final report. They were great in the era of physically printed and bound reports that lived on bookshelves, but in a digital

world, the filename takes the place of the cover page. Your report should document what you built (or tried to build), system architecture, components needed, etc. (Especially for Zynq folks). You should include information about your design process: what challenges did you run into, and (how) did you overcome them? What other libraries did you consider/experiment with and why did you abandon them? In a fast-moving, ever-changing field like embedded systems, knowing why you did something and why other implementations don't work is as (more?) important to document than how you built what you built.

**Journal**

Every day you work on the project, record approximate time spent working on the project, work completed, and issues you encountered. This is an informal document that does not need any formatting. Screenshots, links, raw data, diagrams, and resources are encouraged. For group projects, each member should complete their journal.

**Hand In**

1) One of: your final report, your instructable, a link to your instructable published elsewhere, your journal

2) The source code authored by your team.

3) (Optional) Updated demo video if different than the one in your final presentation

# Appendix C

## RESEARCH TOPIC PRESENTATION DESCRIPTION

Students will choose a current advanced topic in real-time embedded systems for the research presentation and give a 30-minute presentation to the class. Presentations can be done individually or in small groups and occur on Thursdays at the beginning of class. Please select a topic different from what your quarter project is on and have the instructor approve it. The goal for this assignment is for you to learn more about a research interest and to introduce the class to a wide variety of current topics. In your presentation, please include:

- Background about the topic. Why is this topic significant? What are the current challenges or limitations?
- What are recent advancements or changes related to the topic?
- What future work is proposed?
- Reference at least two recent technical published papers related to your topic

# Appendix D

## WEEK 1 LECTURE SLIDES

# Introduction

Week 1

# EE/CPE 442 and EE/CPE 542

- 442 introduced tools, high-performance coding techniques, and custom HW/software interaction
  - Threads, CPU inefficiency, vectorization, compiler optimization, coding style, SIMT, performance counters
- 542 introduces more advanced topics and is more focused on individual projects

# Course Topics

- OS topics
  - Review topics about operating systems, including the differences between operating systems for real-time and embedded purposes and general purpose
- Task scheduling
  - OS task management and scheduling for real-time systems
  - Single and multi-processor
  - Common task and processor scheduling algorithms
- Resource management
  - How to manage resource dependencies between tasks and processors
  - Resource management protocols

# Course Topics

- Inter-process communication
  - How to perform communication between separate tasks and processes
  - Common communication methods
- Buses
  - Bus communication in an embedded system
  - Bus arbitration protocols
- Board I/O
  - Hardware and communication methods for performing input and output on an embedded system
- Final project presentations

# Textbooks

- All textbooks available through the Cal Poly library
- Real-Time Embedded Systems by Jiacun Wang, 2017
  - Topics from this course include tasks scheduling, resource management, OS topics
  - Topics from 442 include intro to RTES and POSIX threads
- Embedded Systems Architecture by Tammy Noergaard, 2013
  - Topics from this course include board I/O and buses
- Multicore DSP by Naim Dahnoun, 2018
  - Topics from this course include inter-process communication

# Quarter Long Project

- Work on an independent or group project over the entire quarter
- The project must:
  - Use at least one non-processor peripheral on your Raspberry Pi or Zybo (DMA engine, graphics processor, SPI interface, etc.)
  - Have a software program running on Linux that controls your overall application
  - Have a means for your application to display results/transfer data/or otherwise interact meaningfully with the outside world
  - Be an innovative, novel, or otherwise comprehensive system worthy of being a quarter-long project
- Project proposal due next Thursday
- See “Final Project Description” on Canvas for more details

# Research Topic Presentation

- Choose a current topic in real-time embedded systems and give a presentation on it
  - Please select something different from what you're interested in doing your quarter project on
  - Once you have a topic, ask the instructor for approval
- Can be done individually or in small groups
- Presentations should include:
  - Background about the topic
  - Any recent advancements or changes
  - Future work
  - At least two technical published papers

# To Do Today

- Start thinking about a topic for the research topic presentation
  - Sign up for a presentation date
- Start thinking about quarter-long projects and groups
  - Project proposal due next Thursday

# Appendix E

## WEEK 2 LECTURE SLIDES

# Operating System

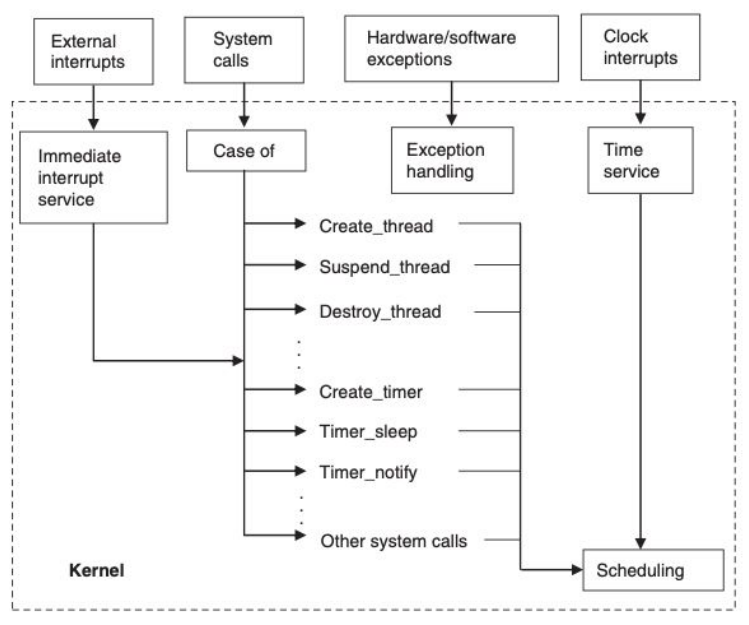
Week 2

# General Purpose OS

- Software that manages hardware resources
- Makes the system more convenient to use
- Includes a kernel and performs functions such as process management, resource management, interrupt management, and I/O management
- Includes an application programming interface (API) that enables applications to communicate with hardware and software applications

# Real Time Embedded System OS

- Embedded systems have limited memory, so the kernel must be smaller and have less functionality (microkernel)
- Real-time systems require the timing behavior of OS to be predictable
  - Must know execution time for OS services
- OS manages the timing and scheduling of tasks
- OS must be fast with little overhead




Structure of a Microkernel

# Microkernel

- Takes control from executing thread to perform a system call, do scheduling and service times, and handle external interrupts
- A system call is a call to one of the API functions (e.g., create/suspend/destroy a thread, create a timer, sleep timer)
- Includes a scheduler responsible for scheduling tasks to meet deadlines (discussed later)
- Handle external interrupts from I/O

# Process Management

- OS is responsible for starting, suspending, and terminating processes
- Processes can be single or multi-threaded
- Each thread requires the OS to allocate memory, create data structures, and copy code
- Threads in the same process use the same address space, global and static variables, code, and heap but have different statuses, program counters, registers, and stacks
  - Threads require fewer resources than separate processes
- OS supports communication between different processes while they're running
  - Inter-Process Communication (IPC) - covered later

# Memory Management

- OS is responsible for all system memory in use
- Generally, only the OS has access to physical memory
  - Memory protection - use virtual memory to ensure separate processes run in their own address space
- When a process starts, the OS allocates memory and loads the process' executable code and initialized data
- OS allocates memory during runtime from the heap (e.g., *malloc* in C)

# Interrupts

- OSs are interrupt-driven
  - When an interrupt occurs, OS transfers control to the Interrupt Service Routine (ISR) to handle the event
- Most modern OSs use split interrupt handling
- Immediate interrupt service
  - Device-dependent
  - Interrupt first assigned a priority
  - Kernel branches to interrupt handling code
  - Interrupt latency - responsiveness of the system to external events
- Scheduled interrupt handling routine
  - Completes interrupt handling
  - Interrupt handling scheduled for execution with other tasks based on priority

# Clocks and Timers

- A precise periodic counter is a hardware clock
- Periodic interrupts from the hardware clock update the kernel's software clock
  - Interrupts every ~100s us to 10s ms
  - Software clock used by threads
- The timer queue stores the expiration times of timers associated with each clock
- Real-time systems often have multiple clocks for different purposes
- Threads and processes can have their own timers bound to a specific clock with an expiration time
- Timer error due to the frequency of hardware clock interrupts and due to the time spent processing timer events

# Clocks and Timers

- The release time of a task may be late due to rescheduling and delays from the time it takes to create a timer
- Periodic tasks are not truly periodic
  - e.g., a loop with a periodic task of 10 ms
  - If the 1st instance of the loop is delayed and finishes after 10 ms, it will block the 2nd instance of the loop

# Examples of RTOS for Embedded Systems

- FreeRTOS
  - Open-source
  - Popular
  - Simple to use
- Zephyr
  - Open-source
  - Secure
  - Simple to use
  - Lots of support
- VxWorks
  - Commercial
  - More safety and security certification, used in defense, medical, and energy industries

## Resources

J. Wang, *Real-Time Embedded Systems*. Hoboken, NJ, USA: Wiley, 2017.

"FreeRTOS Real Time Operating System for Microcontrollers," *FreeRTOS*, 10-Mar-2022. [Online]. Available: https://www.freertos.org/. [Accessed: 29-Apr-2022].

*Wind River*. [Online]. Available: https://www.windriver.com/. [Accessed: 29-Apr-2022].

*Zephyr Project*, 22-Feb-2022. [Online]. Available: https://www.zephyrproject.org/. [Accessed: 29-Apr-2022].

# Appendix F

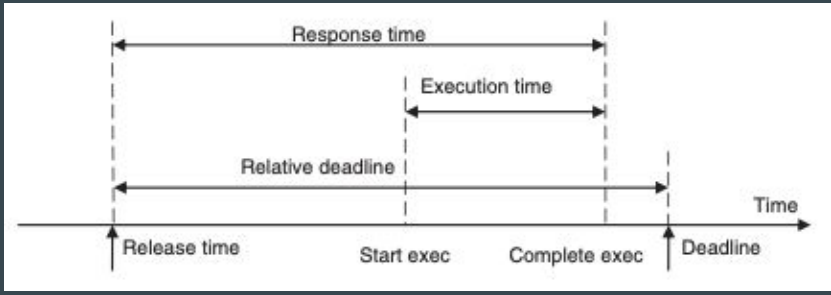
## WEEK 3 LECTURE SLIDES

# Task Scheduling

Week 3

# Tasks

- RTOS kernel has a scheduler that allocates and schedules tasks
- Task - a unit of work scheduled for execution on the CPU
  - Release time - the time the task becomes available for execution
  - Deadline - the time execution must be completed by
  - Relative deadline - the deadline relative to the release time
  - Execution time - the time it takes for a task to execute when it executes alone and with all required resources
  - Response time - the time from a task being released to its execution completing



# Types of Deadlines

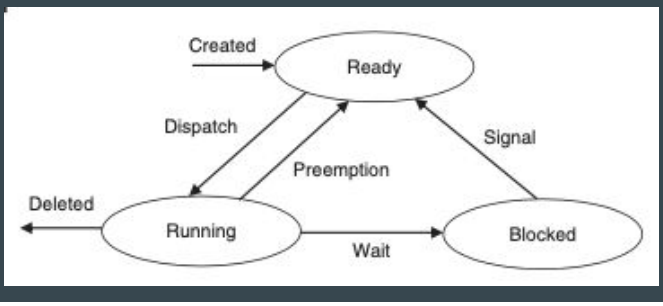
- Hard - the task must be completed by a specific time
  - if the deadline is missed, the system fails, or the task has no use
  - Example: when a collision is detected, a cards airbags must deploy within 60 ms
- Soft - the task should be completed by a specific time
  - if the deadline is missed, there aren't any disastrous results, and the task still has some use
  - Example: after a debit card is inserted, the ATM should prompt the user for a PIN within 1 s

# Task States

- Running - the state when the task is executing
  - Only one task can run on a processor at a time
- Ready - a task can't execute because another task is running
  - Has all other resources except the processor
- Blocked - a task is waiting for a time or external event to occur
  - Not available for scheduling

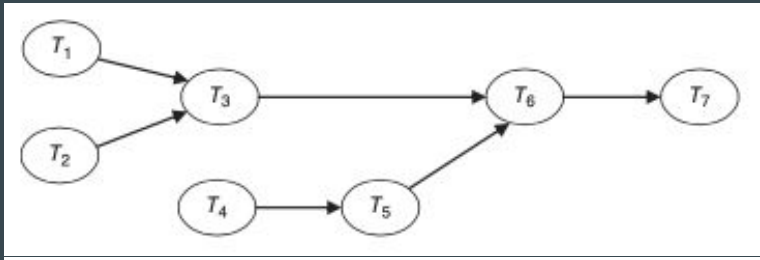
# Task States

- New tasks are placed in the ready state queue and scheduled for execution based on their priority
- When the highest priority task is dispatched for execution, it shifts to the running state
- A task in the running state can be preempted (suspended) by a higher priority task and placed in the ready queue
- Running task can be moved to a blocked state (e.g., due to memory access issues)



# Precedence

- Defines execution order of 2+ tasks
- Shows data and control dependencies among tasks
- Use a graph to show precedence




$T_1 < T_3, T_2 < T_3, T_3 < T_6, T_4 < T_5, T_5 < T_6,$ and $T_6 < T_7$.

# Preemptivity

- Which task the processor is executing can change during runtime while a task is executing
- e.g., the scheduler may suspend the execution of a task for a more urgent task to run
  - The original task resumes once the more urgent task completes
- A task is preemptable if it can resume its execution from the point of interruption
- A task is non-preemptable if it must execute without interruption
- A task is partially preemptive if only a section of the task is non-preemptable, but the rest of the task is preemptable

# Multiprocessor Task Assignment

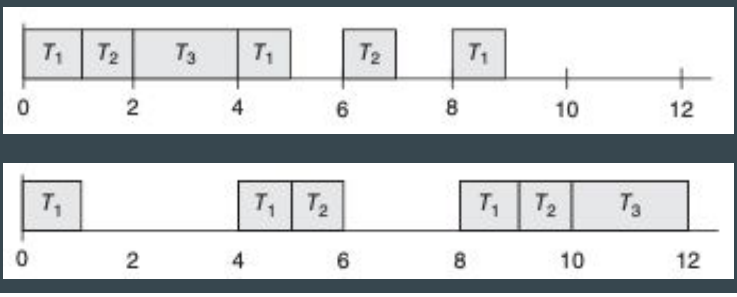
- Multiprocessor scheduling - tasks are assigned to processors, then perform uniprocessor scheduling for tasks on each processor
- Redo task allocation or scheduling if the schedule is infeasible
- All of the following algorithms are for uniprocessor scheduling

# Clock Driven Scheduling

- Scheduling decisions are made at specific time instants
  - Either random or periodic time points
- Works for deterministic systems - tasks have hard deadlines, and task parameters don’t change
- Can be computed and stored before runtime
  - Saves runtime scheduling and overhead

# Clock Driven Scheduling - Periodic Tasks

- Repeated once a period
  - Have a hard deadline
- Consider 3 tasks to be completed in 12 time units
  - T1: deadline = 4, execution time = 1, 1 instance
  - T2: deadline = 6, execution time = 1, 2 instances
  - T3: deadline = 12, execution time = 2, 1 instance
- How can all tasks be scheduled between 0 and 12 time units?
- Scheduling can be performed at random time intervals

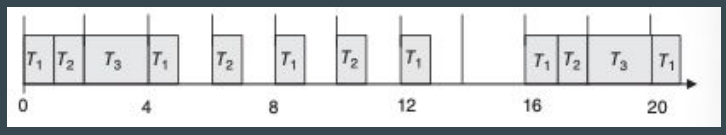


# Structured Clock Driven Scheduling

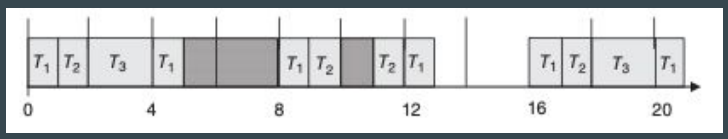
- Scheduling decisions made at periodic times
  - Periodic scheduling creates intervals called frames
- Scheduling decisions made at the beginning of the frame
- Frames need to be big enough to allow all tasks to complete execution
  - At least one full frame between the arrival of a task instance and its deadline
- Can slice larger tasks into smaller ones to allow them to complete within one frame

# Clock Driven Scheduling - Aperiodic Tasks

- One-shot tasks caused by external events
  - No deadline or soft-deadline
- Scheduled to fit in idle spots (slack) between periodic tasks
- Slack stealing - delay the execution of periodic tasks so aperiodic tasks can execute earlier




Periodic tasks scheduled for execution



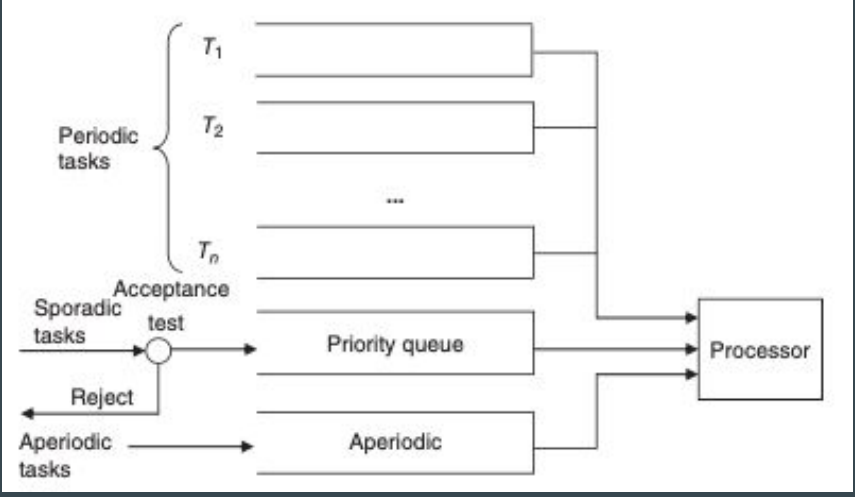

Periodic and aperiodic tasks scheduled for execution with slack stealing

# Clock Driven Scheduling - Sporadic Tasks

- Event-driven and have an unknown arrival time
  - Have a hard deadline
- Release time, execution time, and deadline aren't known in advance
  - No way to guarantee they meet their deadline
- Scheduler tests if it can schedule a sporadic task and meet its deadline
  - Rejects task if it fails

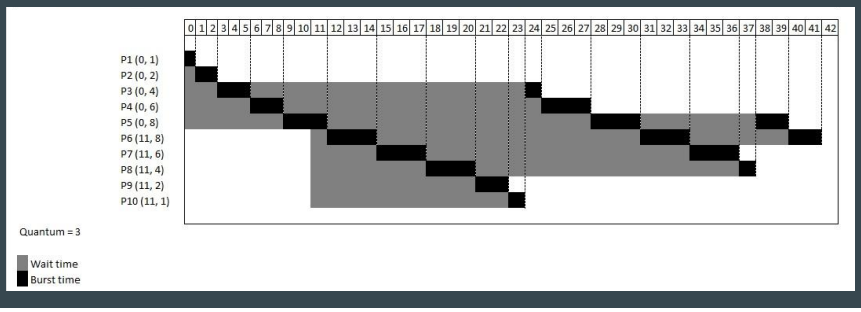
# Clock Driven Scheduling

- Place all aperiodic tasks in the aperiodic queue and all sporadic tasks in the priority queue
  - Priority queue - tasks with the earliest deadline are at the head of the queue



# Round-Robin Approach

- First-in-first-service (FIFS) queue that stores all tasks in the ready state
- The task at the head of the queue is removed, and it executes during a short time slice
- If a task doesn't finish execution in the time slice, its placed at the tail of the FIFS queue
- May cause tasks to miss their deadlines
- Weighted round-robin - higher priority tasks get longer time slices



# Resources

J. Wang, *Real-Time Embedded Systems*. Hoboken, NJ, USA: Wiley, 2017.

# Appendix G

## WEEK 4 LECTURE SLIDES

# Task Scheduling

Week 4

# Priority-Driven Scheduling

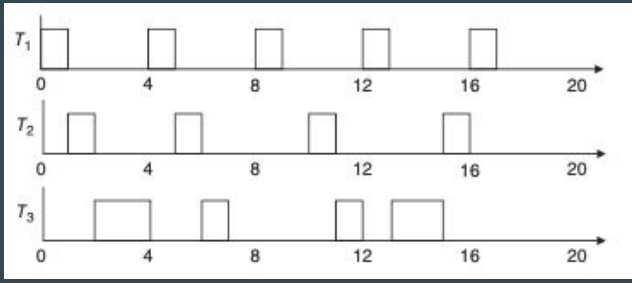
- Scheduling decisions made when a new task is released or completed
- Scheduling is done at run-time
  - Doesn't require information about release and execution times before run-time
  - Required for systems whose workload is unpredictable
- Priority is assigned to each task

# Priority-Driven Scheduling - Rate Monotonic (RM) Algorithm

- Assign priority to tasks based on their period
  - Shorter period = high priority
- If a task is released and the processor is idle, it executes the task
- If there's another task running, it compares the priority of the two tasks
- If the new task's priority is higher, then it preempts the running task and executes

# Priority-Driven Scheduling - RM Algorithm Example

- Consider 3 tasks
  - T1: deadline = 4, execution time = 1, highest priority
  - T2: deadline = 5, execution time = 1, middle priority
  - T3: deadline = 10, execution time = 3, lowest priority
  - All tasks complete in time

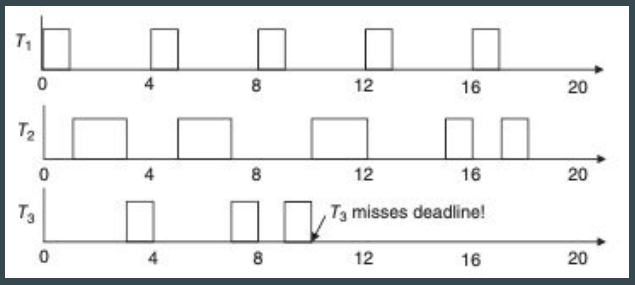



T3 cannot execute when T1 or T2 is unfinished

T1 preempts T3 at time 4 and 12

# Priority-Driven Scheduling - RM Algorithm Example

- Consider 3 tasks
  - T1: deadline = 4, execution time = 1, highest priority
  - T2: deadline = 5, execution time = **2**
  - T3: deadline = 10, execution time = **3.1**
  - Task 3 misses its deadline at 10 because it doesn't have time to execute when tasks 1 and 2 are idle




T3 cannot execute when T1 or T2 is unfinished

T1 preempts T3 at time 4 and 8

T2 preempts T3 at time 10

T3 still has 0.1 execution time left at its deadline

# Deadline Monotonic Algorithm

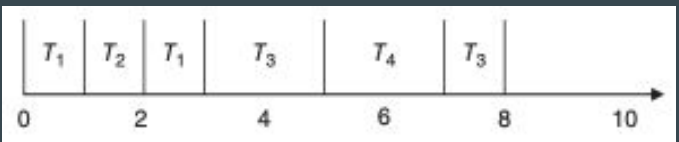
- Assigns priorities to tasks based on their relative deadlines
  - Shorter relative deadline = higher priority

# Earliest Deadline First (EDF)

- Assigns priority to tasks based on their absolute deadline
  - Earlier absolute deadline = higher priority
- Priority decided at runtime
- Optimal scheduling algorithm - if a set of tasks has a feasible schedule, EDF can produce a feasible schedule

# EDF Example

| Task | Release time | Execution time | Deadline |
|---|---|---|---|
| $T_1$ | 0 | 2 | 6 |
| $T_2$ | 1 | 1 | 4 |
| $T_3$ | 3 | 3 | 10 |
| $T_4$ | 5 | 2 | 8 |




T2 preempts T1 at time 2 because it has an earlier deadline

T4 preempts T3 at time 5 because it has an earlier deadline

# Priority Driven Scheduling - Sporadic Tasks

- Sporadic tasks have a minimum interarrival time between any two instances of the task
- Sporadic tasks can be treated as periodic tasks with a period equal to interarrival time
- Can use a polling server similar to aperiodic tasks
- Perform an acceptance test to check if the task will meet the deadline before scheduling it

# Priority Driven Scheduling - Aperiodic Tasks

- Aperiodic tasks have the lowest priority
  - No guarantee of their response time
- Execute during slack times of periodic tasks
  - Slack stealing - delay the execution of periodic tasks so aperiodic tasks can execute earlier
  - Slack stealing is more complicated with priority-driven scheduling over clock-driven scheduling because scheduling decisions are made at runtime
- Use a polling server
  - The polling server is treated as a periodic task and has priority based on its polling period
    - The polling period is the server's execution time
  - When the polling server executes, it executes the task at the head of the queue
  - If no tasks are available for execution, the polling server suspends itself

# Non-preemptivity

- A task or a portion of a task may be non-preemptable
  - A task may be running in a critical section
  - Preemption is too costly
- A non-preemptable task may block a higher priority task from executing
- Must be considered when testing the schedulability of a task

# Self-Suspension

- Task can self-suspend if it's waiting for an external operation to complete on another processor
- Task loses processor and is placed in a blocked queue by the scheduler
- Can delay execution of lower priority tasks

# Context Switches

- Context - data indicating execution status and stored in the task control block (TCB)
  - TCB - a data structure that contains information about the execution of the task
- When a task is switched out of the CPU, its context must be stored
- When its switched back into the CPU, its context must be restored so it can execute from the last point
- Can add context switch time to execution time of task instances
  - CS = max time for context switch
  - Without self-suspension, add 2CS to the execution time of each task
  - With self-suspension, add 2(k+1)CS to execution time, where k is the number of self-suspensions

# Processor Assignment

- All previous algorithms were for uniprocessor
- For multiprocessors, the tasks must first be assigned to a processor before uniprocessor scheduling can occur
- When assigning tasks to processors, we need to consider task execution times, communication costs between tasks, and placement of resources

# Bin-Packing Algorithms

- Doesn’t take communication costs into consideration
- Put tasks on a finite number of processors such that the number of processors is minimized
  - Leave some processor space for the execution of aperiodic and sporadic tasks

# First-Fit Algorithm

- Processes tasks in random order
- Places task on the first processor that can accommodate the task
- If there's no available processor, it adds the task to a new one
- This algorithm uses less than twice the optimal number of processors given a set of tasks
- First-fit decreasing algorithm - tasks are first sorted in decreasing order of utilization (execution time/period)

# Resources

J. Wang, *Real-Time Embedded Systems*. Hoboken, NJ, USA: Wiley, 2017.

# Appendix H

## WEEK 5 LECTURE SLIDES

# Resource Management

Week 5

# Purpose of Resource Management

- Tasks have resource dependencies among them
- Previous scheduling algorithms ignored resource dependencies between tasks
- Shared resources don't allow simultaneous access
- Common resources that must be shared are data structures, variables, main memory, files, registers, and I/O units

# Clock vs Priority Driven Scheduling

- Clock-driven systems don’t have scheduling issues due to shared resources
  - The scheduler can schedule tasks to keep data serialized
- Priority-driven systems are affected by shared resources
  - Resource management applies to priority-driven systems

# Resource Locking

- When a task requests to use a resource, it locks that resource and unlocks it when it's done with the resource
- Lock request fails if the scheduler doesn't grant resources to the task
  - The task becomes blocked and losses the processor
  - Removed from the ready queue and stays blocked until the scheduler grants resources
  - Becomes unblocked and is moved to the ready task queue

# Priority Inversion

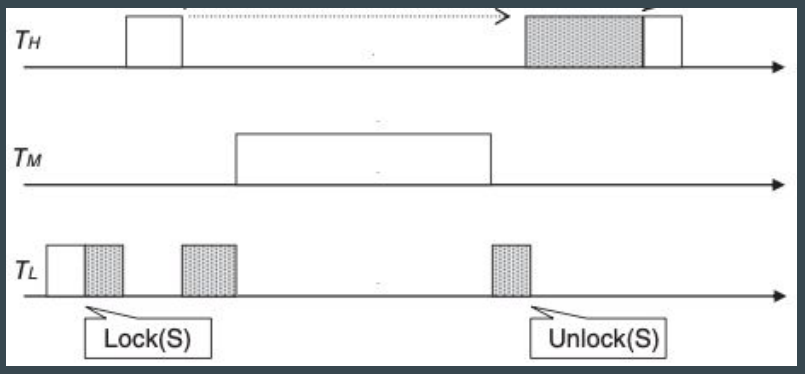
- When a higher and lower priority task share a resource and the lower priority task is using the shared resource
- The higher priority task must wait for the lower priority task to finish
- Priority inversion - a lower priority task is running while a higher priority task is waiting
    - Bounded - as long as the lower priority task doesn't take too long, the higher priority task can still meet its deadline

# Priority Inversion

- If a medium priority task without a shared resource is released it will preempt a lower priority task
- The low priority task must wait for the medium priority task to finish
  - The high priority task must wait for the medium and low priority task to finish
- If a second medium priority task preempts the first medium priority task, it can create a chain of waiting tasks
- Unbounded priority inversion - high priority task is blocked due to resource access and miss its deadline

# Priority Inversion

- Unbounded priority inversion
- Low priority task locks resource S, and medium priority task preempts it
- The high priority task must wait for medium and low priority tasks to finish to access resource S and execute
- The high priority task may miss its deadline

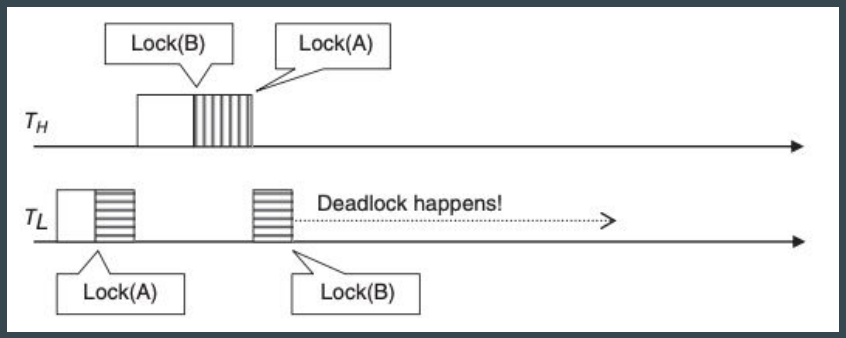


# Deadlock

- State where no tasks make progress due to accessing shared resources
- Happens when:
  - A task exclusively holds one or more resources
  - A task holds a resource while waiting for another resource
  - Resources aren't preemptable
  - Circular chain of tasks where each task is waiting for a resource held by the next task in the circle
    - e.g., consider tasks T1, T2, and T3, T1 needs T2's resources, T2 needs T3's resources, T3 needs T1 resources

# Deadlock

- Low priority task locks resource A
- High priority task preempts low priority task and locks resource B
- High priority task tries to lock resource A but is blocked
- Low priority task executes and tries to lock resource B but is blocked
- Both tasks are blocked, and neither can execute



# Resource Access Control Protocols

- Needed to regulate access to shared resources
- Needed to handle priority inversion and deadlocks caused by resource sharing
  - Can prevent deadlocks
  - Can’t remove all priority inversion but can decrease the blocking time of high priority tasks
- Set of rules that determine when requests for resources are granted and how tasks requiring shared resources are scheduled

# Non-preemptive Critical Section Protocol

- Unbounded priority inversion significantly hurts applications
  - The goal is to prevent unbounded priority inversion
- When a task lacks a resource, it executes at a higher priority that's higher than the priority of other tasks until it unlocks the resource
  - Task can't be preempted when it holds a resource
  - Prevents circular waiting
- A higher priority task can only be locked once (by a lower priority task that it shares a resource with)

# Non-preemptive Critical Section Protocol

- Simple and easy to implement
- Don’t need prior knowledge of resource requirements of tasks
- Eliminates unbounded priority inversion and deadlocks
- Doesn’t reduce bounded priority inversion

# Priority Inheritance Protocol

- Eliminates unbounded priority inversion but not deadlocks
  - Doesn't eliminate circular waiting for resources
- When a lower priority task blocks a higher priority task, it inherits the priority of the blocked higher priority task
  - Task returns to original priority after releasing its resource
- Assigned priority - priority assigned by the scheduling algorithm
- Current priority - priority inherited from another task

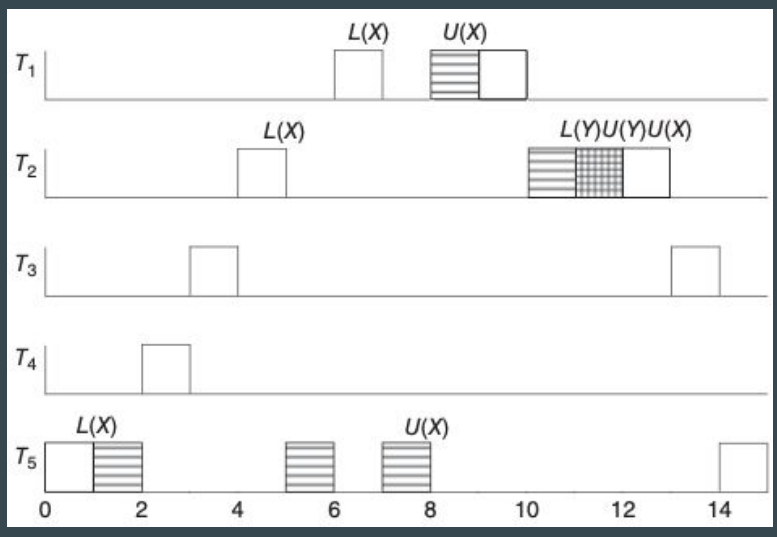
# Priority Inheritance Protocol

- Ready tasks are scheduled according to their current priorities
  - The current priority is the same as a task's assigned priority unless it's using a shared resource
- When a task locks a resource,
  - If the resource is free, then it's allocated to the task ,and the lock is successful
  - If the resource isn't free, then the lock is denied, and the task is blocked
- When a lower priority task blocks a higher priority, it inherits the current priority of the blocked higher priority task until it unlocks the resource
- A task can be blocked by many lower priority tasks, causing the first task to miss its deadline
- A task can be blocked by a lower priority task that it has no resource conflict with

# Priority Inheritance Protocol

- T1 is highest priority, T5 is lowest priority
- T5 locks resource A at time 1
- T4 preempts T5,  then T3 preempts T4, then T2 preempts T3
- T2 attempts to lock resource A at time 5
  - T2 is blocked by T5
  - T5 inherits T2's priority and executes
- T1 preempts T5 at time 6

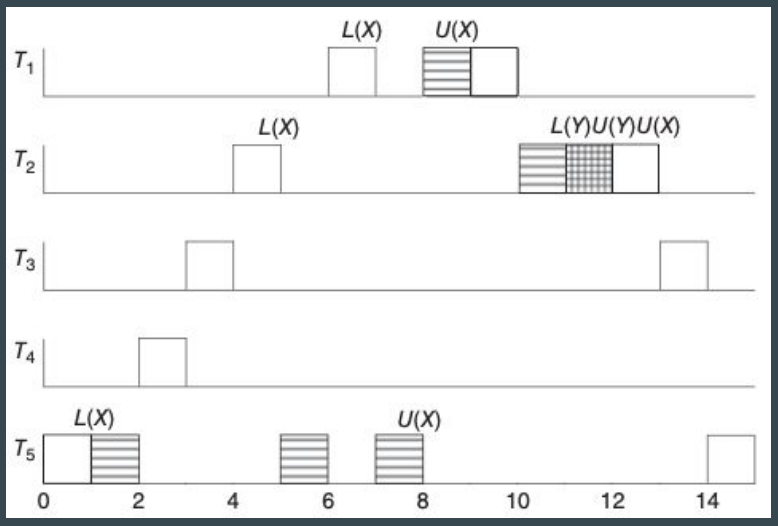
| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $X$; [2, 3) uses $Y$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Priority Inheritance Protocol

- T1 attempts to lock resource A at time 7
  - T1 is blocked by T5
  - T5 inherits T1's priority
- T5 unlocks resource A
  - Current priority decreases to assigned priority
- T1 preempts T5 and is granted A
- T1 unlocks resource A and completes execution
- T2, then T3, then T5 complete execution

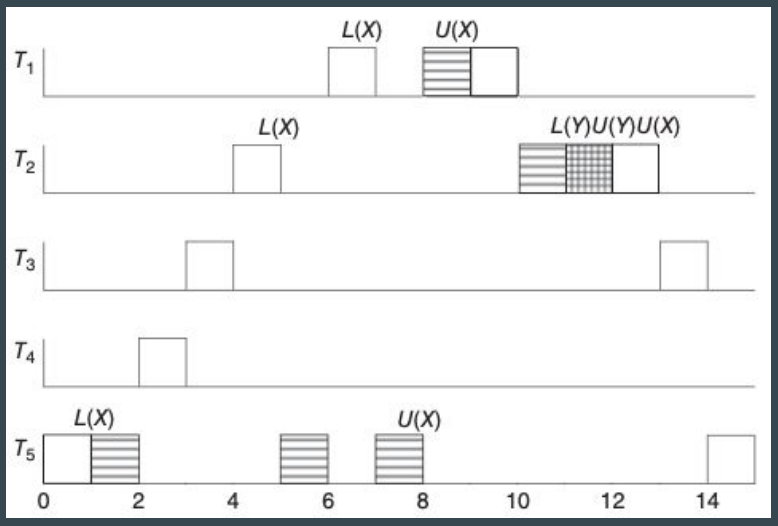
| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $X$; [2, 3) uses $Y$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Priority Inheritance Protocol

- Prevents unbounded priority inversion because T3 doesn't preempt T5
- T1 and T2 are blocked by T5 due to resource sharing

| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $X$; [2, 3) uses $Y$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Resources

J. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2009.

J. Wang, *Real-Time Embedded Systems*. Hoboken, NJ, USA: Wiley, 2017.

# Appendix I

## WEEK 6 LECTURE SLIDES

# Resource Management

Week 6

# Priority Ceiling Protocol

- Similar to priority inheritance protocol
- Avoids unbounded priority inversion
- Helps avoid deadlocks but doesn't prevent them
- Assumes resource requirement of tasks is known before execution
- Associates each resource with a priority ceiling
  - Priority ceiling - highest priority of all tasks that might use that resource
- Request for a resource is only successful if the priority ceiling of the new resource is higher than any preempted resources

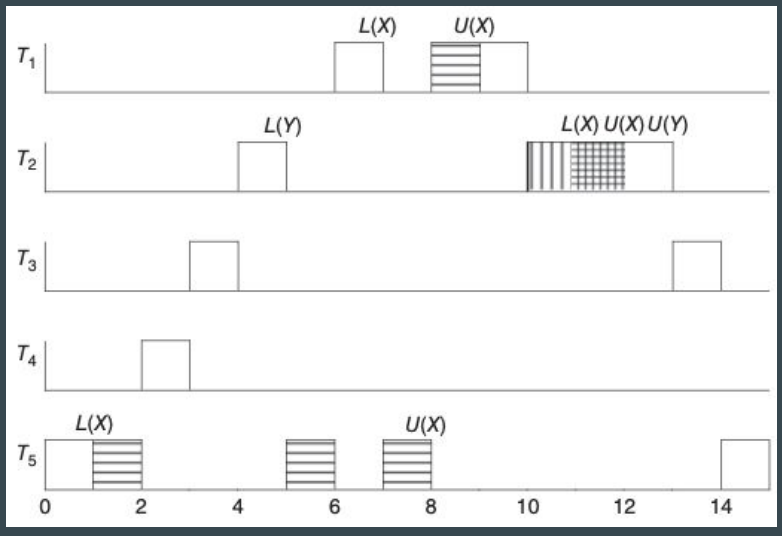
# Priority Ceiling Protocol

- A task's current priority is its assigned priority unless it's blocking a higher priority task
- System's priority ceiling - highest priority ceiling of all resources in use
- When a task requests a resource
  - If the resource isn't free, the lock is denied, and the task is blocked
  - If the current priority is higher than the system's priority ceiling, then the lock is successful
  - If the current priority is lower than or equal to the system's priority ceiling, the lock is success if the task is the task holding the resource whose priority ceiling is equal to the system's priority ceiling
    - If not, the lock is denied, and the task is blocked
- When a lower priority task blocks a higher priority task, it inherits the current priority of the blocked task until it unlocks the resource

# Priority Ceiling Protocol

- Priority ceiling for resource X = 2
- Priority ceiling for resource Y = 1
- T5 locks resource X at time 1
- T5 is preempted by T4, T4 is preempted by T3, T3 is preempted by T2
- T2 requests resource Y at time 5
  - Priority of T2 is less than the priority ceiling so T2 is blocked by T5
  - T5 has a current priority of 2
- T1 preempts T5

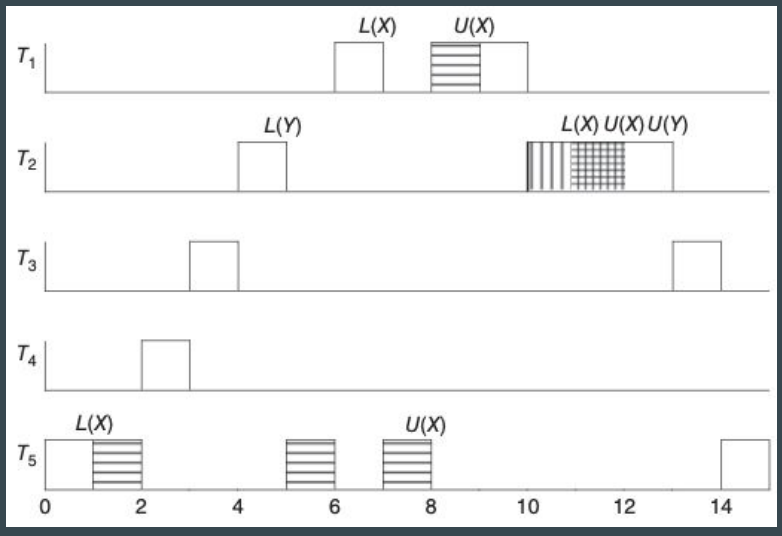
| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $Y$; [2, 3) uses $X$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Priority Ceiling Protocol

- T1 requests resource X at time 7
  - Blocked by T5
  - T5 has a current priority of 1
- T5 unlocks X and T1 is unblocked
- T1 unlocks X
- T2 executes and requests resource X
  - T2's priority equals the priority ceiling
  - T2 locks resource X
- T2, T3, and T5 complete execution

| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $Y$; [2, 3) uses $X$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Priority Ceiling Protocol

- Worst case blocking time is the time that a lower priority task has a resource locked
- Doesn't block tasks that don't use shared resources

# Stack-Sharing Priority Ceiling Protocol

- Simplifies priority ceiling protocol
- No unbounded priority inversion
- No deadlocks
- No chained blocking
- Uses stack sharing among tasks to reduce overhead due to context switches
  - Normally each task has its own runtime stack to store local variables and return addresses
- Executing task is on top of the stack
  - Task removed from the stack when it completes execution

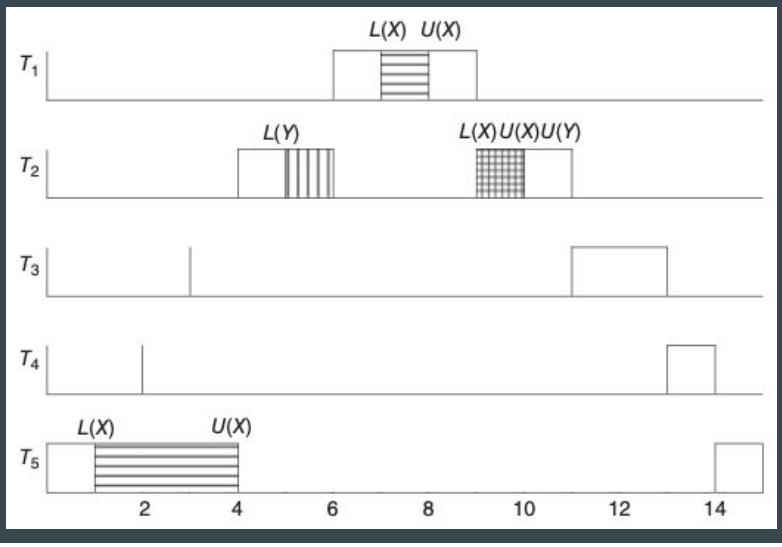
# Stack-Sharing Priority Ceiling Protocol

- Need to make sure that an executing task doesn't get blocked due to shared resources
  - That would automatically lead to deadlock because executing task can't be removed from the top of the stack
  - Blocking can only occur when a task is first released
- The system's ceiling priority is updated each time a resource is allocated or freed
- After a task is released, its blocked from executing until its assigned priority is greater than the system's ceiling priority
- When a task requests a resource, it's allocated the resource

# Stack-Sharing Priority Ceiling Protocol

- T5 is released and it executes
- T5 locks resource X
  - System's ceiling priority is updated to 1
- T4 and T3 are released and blocked by T5
- T5 unlocks X at time 4
  - System's ceiling priority is updated to 0
- T2 is released and preempts T5
- T2 locks resource Y
  - System's ceiling priority is updated to 2

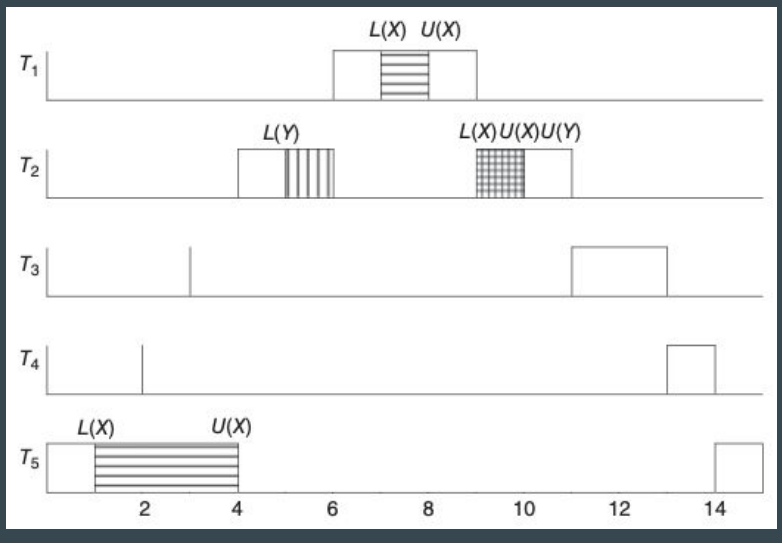
| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $Y$; [2, 3) uses $X$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Stack-Sharing Priority Ceiling Protocol

- T1 preempts T2
- T1 locks resource X
  - System's ceiling priority is updated to 1
- T1 unlocks resource X and completes execution
  - System's ceiling priority is updated to 2
- T2 executes and locks resource X
  - System's ceiling priority for X is updated to 1
- T2 unlocks resources X and Y and completes execution
  - System's ceiling priority is updated to 0

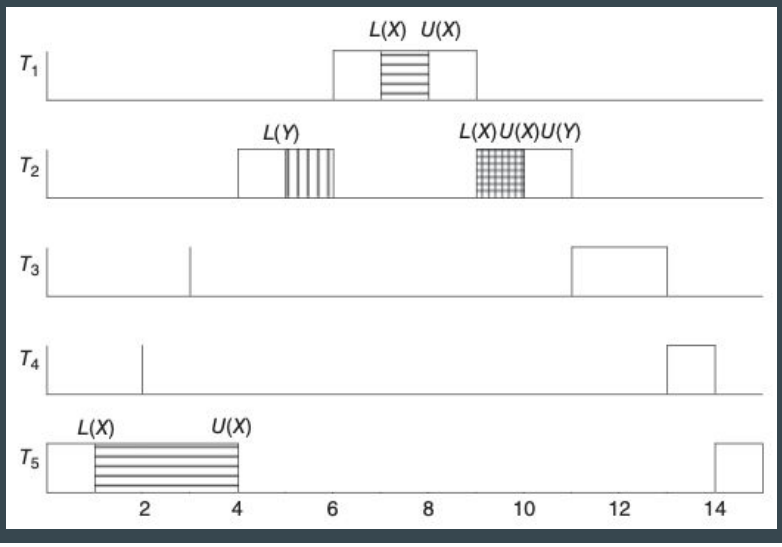
| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $Y$; [2, 3) uses $X$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Stack-Sharing Priority Ceiling Protocol

- T3 has the highest priority and completes execution
- T4 has the highest priority and completes execution
- T5 completes execution

| Task | Priority | Released at | Execution time | Resource usage |
|---|---|---|---|---|
| $T_1$ | 1 | 6 | 3 | [1, 2) uses $X$ |
| $T_2$ | 2 | 4 | 5 | [1, 3) uses $Y$; [2, 3) uses $X$ |
| $T_3$ | 3 | 3 | 2 | None |
| $T_4$ | 4 | 2 | 1 | None |
| $T_5$ | 5 | 0 | 5 | [1, 4) uses $X$ |



# Resource Contention with Aperiodic Tasks

- Modify aperiodic and sporadic scheduling algorithms
- The polling server is non-preemptable when the aperiodic task has a locked resource
  - Task can run past scheduled execution time for polling server
- If the task ran beyond the polling server period, the overrun time is subtracted from the next instance of the polling server

# Resources

J. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2009.

J. Wang, *Real-Time Embedded Systems*. Hoboken, NJ, USA: Wiley, 2017.

Appendix J

WEEK 7 LECTURE SLIDES

# Inter-Process Communication

Week 7

# Purpose of Inter-Process Communication

- Need to exchange data between threads and synchronize them
- Real-time applications need high bandwidth and low latency
- Used for data sharing between processes on a single core or different cores

# Semaphore

- Counting semaphore - stores a value initialized to 0 that can be incremented
- Binary semaphore - special case semaphore that stores 0 or 1
- Mutex - stores a 0 or 1
    - Only one task or process can acquire the mutex

## Shared File

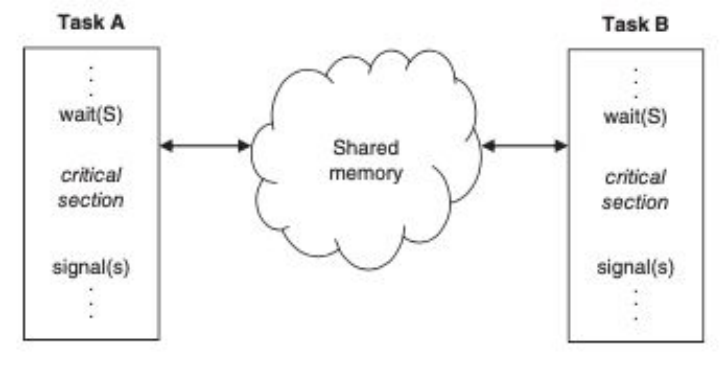
- Most basic method
- Can share large amounts of data
- File access is slow
- The sender creates and writes to a file, and the receiver reads from the same file
- Can create a race condition if the file isn't locked
- The sender needs an exclusive lock so only that process can access the file until it unlocked
- The receiver needs a lock so the file can't be edited while it's reading

# Shared Memory

- Fast and simple communication mechanism
  - Minimizes processing and storage overhead
- All cores have access to device memory
- Senders and receivers can communicate by exchanging pointers
  - The sender writes data to a specific memory location, notifies the receiver, and sends the data pointer to the receiver
  - The receiver can access memory and notifies the sender when it's finished with data
- Only uses load and store instructions
  - Doesn't require OS
- Need synchronization mechanism to make sure processors don't overwrite data
  - Semaphore

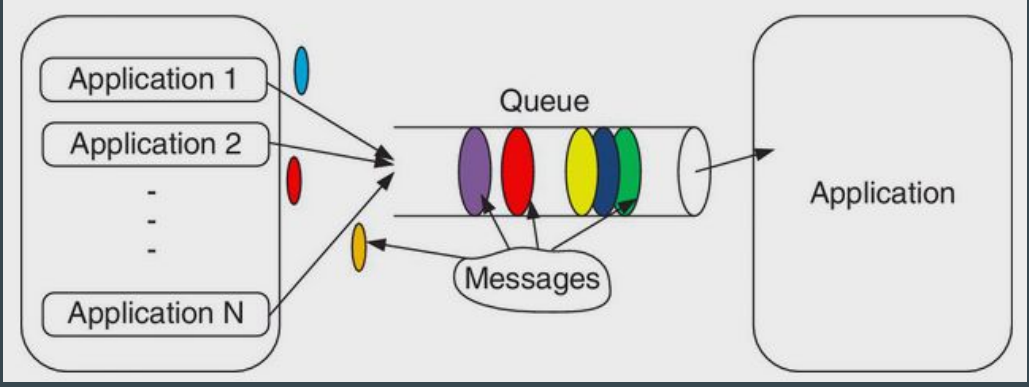
# Shared Memory

- Using shared memory can lead to priority inversion, where a low priority task blocks a high priority task because the low priority task is accessing shared memory



# Message Queue

- The sender writes a message to a message queue, and the receiver reads from the queue
- Doesn't require synchronization
- Easy to manage with lots of processors
- Multiple queues can be used for different data types



# Unnamed Pipes

- For communication between parent and children processes
  - Can't use for unrelated processes
- One process writes to the pipe, and the other reads from it
  - One file is used to write into and another to read from
- The pipe disappears once it's closed or if either of the processes terminates
- One-way communication
  - Need two pipes for bi-directional communication
- Has little overhead

# Unnamed Pipe Commands

- System call to create an unnamed pipe, fd are file descriptors for reading and writing

*int pipe(int fd[2]);*

- Write to the pipe by specifying file descriptor, data, and size of data

*write(fd[1], “Hello!”, 7);*

- Read from the pipe by specifying file descriptor, buffer, and size of data

*read(fd[0], buf, 7);*

# Named Pipes

- Similar to an unnamed pipe but has a file name that's stored in memory and can be accessed by different processes
- Can be used for related or unrelated processes
- Bi-directional communication
- Still exists if one of the processes is terminated
    - Has to be explicitly deleted

# IPC Socket

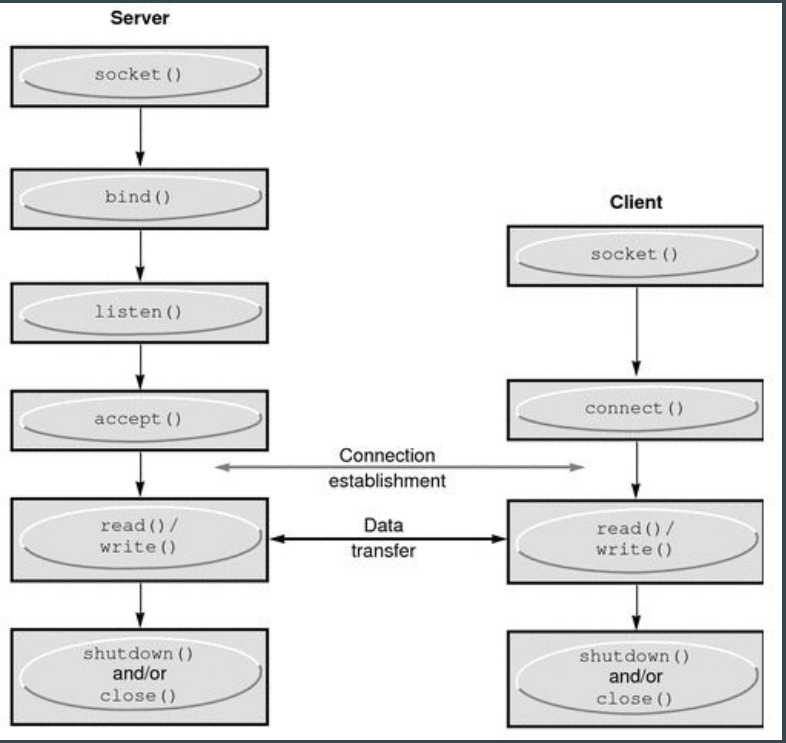
- Bi-directional communication
- IPC sockets use the kernel to support communication between a client and server
- Two processes communicating each have a socket
- A socket address is a network address with a port number
- One process acts as the server and the other as the client
  - The server listens to a specific port and waits for client's request
  - Accepts connection from the client socket
- Processes can communicate until the channel is closed on either end

# Stream Socket

- Requires servant and client to establish a connection first
- Data transmitted in a stream of bytes
- Sequenced - data packets are received in the order they're sent
- Unduplicated - data packets aren't duplicated
- Doesn't keep records of data sent
- Can handle large amounts of data
- Reliable data transfer

# Stream Socket Creation

- The server creates a socket, binds a name to it, and displays the port number
- The client creates a socket and requests a connection to the server
- The server accepts, and the client and server begin communication

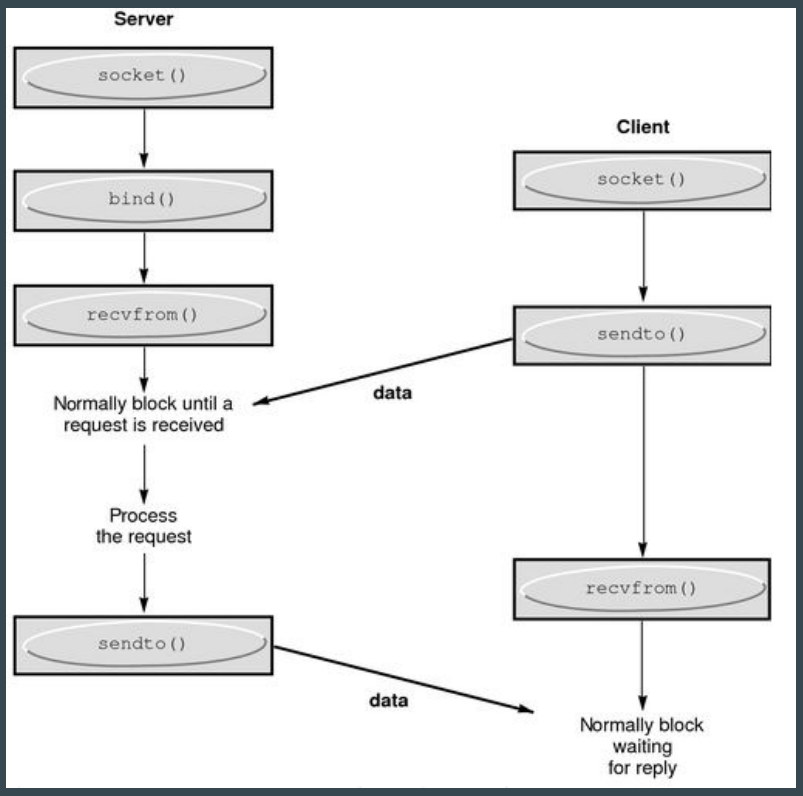


# Datagram Socket

- Not guaranteed to be sequenced or unduplicated
- Keeps records of data sent
- Less overhead than streaming sockets
- Unreliable - data can fail to arrive
  - More reliable for local networks, less reliable over the internet
- Doesn’t require explicit connection
  - Each packet is individually addressed to the socket address
  - Packets can be sent on different routes
- Works better for record-oriented data

# Datagram Socket

- The server creates a socket and binds a name to it
- The client creates a socket
- The server and client can send information to each other use each other's addresses



# Resources


A. De George, “Memory-Mapped Files,” *Memory-Mapped Files | Microsoft Docs*, 15-Sep-2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files. [Accessed: 27-Apr-2022].

M. Kalin April 15, “Inter-process communication in linux: Shared storage,” *Opensource.com*, 15-Apr-2019. [Online]. Available: https://opensource.com/article/19/4/interprocess-communication-linux-storage. [Accessed: 27-Apr-2022].

M. Kalin April 16, “Inter-process communication in linux: Using pipes and message queues,” *Opensource.com*, 16-Apr-2019. [Online]. Available: https://opensource.com/article/19/4/interprocess-communication-linux-channels. [Accessed: 27-Apr-2022].

M. Kalin April 17, “Inter-process communication in linux: Sockets and signals,” *Opensource.com*, 17-Apr-2019. [Online]. Available: https://opensource.com/article/19/4/interprocess-communication-linux-networking. [Accessed: 27-Apr-2022].

N. Dahnoun, *Multicore DSP*. Hoboken, NJ: Wiley, 2018.

T. Whitney, “Windows Sockets: Datagram sockets,” *Microsoft Docs*, 08-Aug-2021. [Online]. Available: https://docs.microsoft.com/en-us/cpp/mfc/windows-sockets-datagram-sockets?view=msvc-170. [Accessed: 27-Apr-2022].

T. Whitney, “Windows Sockets: Stream Sockets,” *Microsoft Docs*, 03-Aug-2021. [Online]. Available: https://docs.microsoft.com/en-us/cpp/mfc/windows-sockets-stream-sockets?view=msvc-170. [Accessed: 27-Apr-2022].

# Appendix K

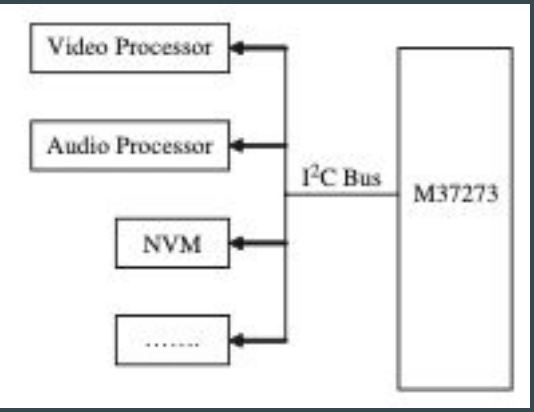
## WEEK 8 LECTURE SLIDES

# Buses

Week 8

# Use for Buses

- A bus is a collection of wires carrying data signals, addresses, and control signals
- All major components that make up an embedded board (processors, I/O, memory) are connected via buses
- Bridges connect buses and carry information from one bus to another
    - Needed for boards with multiple buses that need to intercommunicate



# Types of Buses

- System bus - connect external main memory and caches to the main CPU
  - Short, high speed, and custom
- Backplane bus - also connect external main memory and caches to main CPU and I/O
  - Not as fast as a system bus
- I/O bus - connects remaining components on board to the main CPU, each other, and I/O
  - Standardized
  - Can be short and high speed or long and slow, depending on the protocol used
  - Can handle interrupt request signals

# Types of Buses

- Expandable - additional components can be plugged into the board and can communicate over the bus to other components
  - e.g., USB, PCI, SCSI
  - More flexible
  - More expensive to implement
  - Board performance can be negatively impacted by adding too many components onto the expandable bus
- Non-expandable - additional components can't be plugged into the board
  - e.g., I2C, DIB, VME

# Bus Arbitration

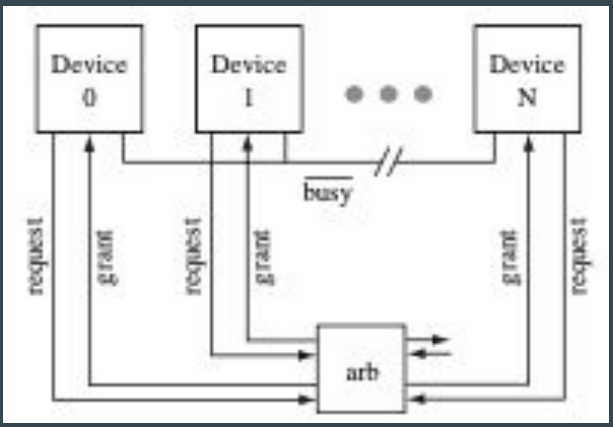
- Buses need protocols that define how devices gain access to the bus, use the bus, and the signals associated with various bus lines
- Main device - the device that initiates a bus transaction
- Secondary device - device that can only gain access to a bus in response to a main device's request
- Only one main and one secondary can communicate over the bus at any given time

# Bus Arbitration

- When there's only one main component and all other components are secondary, no arbitration is needed
- Need arbitrator when there's more than one main device
  - The arbitrator determines which main device gets control of the bus
- Arbitrators can grant a bus to the main device until the main device is finished with its transmission, or the arbitrator can preempt devices in the middle of a transmission

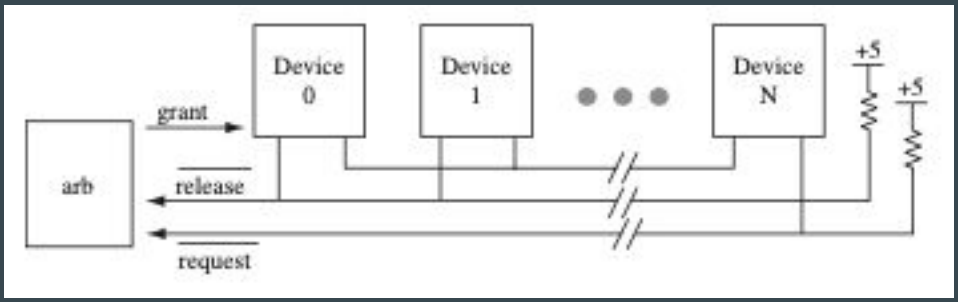
# Dynamic Central Parallel Bus Arbitration

- The arbitrator is centrally located and connected to all main devices
- Mains granted access based on first-in-first-out (FIFO) or priority-based system
- For FIFO, there's a possibility of a single main device maintaining control of the bus and never completing



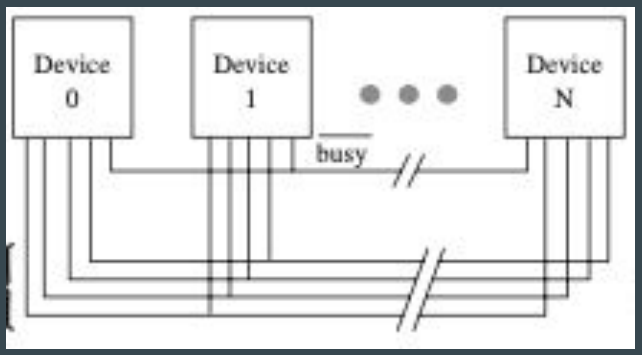
# Centralized Serial Bus Arbitration

- Arbitrator connects to all main devices, and all main devices are connected in series
- The first main device in the chain is granted the bus and passes it to the next in the chain if the bus is no longer needed



# Distributed Self-Selection Bus Arbitration

- No central arbitrator or additional circuitry
- Mains share priority information to determine if a higher priority main is requesting the bus
- Remove arbitration lines and see if there's a collision on the bus



# Bus Timing Scheme

- Synchronous bus - includes clock signal along with data, address, and other control information
  - All devices run at the same clock rate, and data is transmitted either on rising or falling edge
  - Faster clock for shorter buses  and slower clock for longer buses
- Asynchronous bus - doesn't include clock signal but transmits other signals for request and acknowledgement
  - More complicated
  - The length of the bus and the number of components communicating over the bus doesn't matter

# Handshake

- Rules that a device needs to follow to complete a transaction (read or write)
  - Varies for different busses
- Generally, handshakes follow a similar process:
  - The main device requests a transaction
  - The secondary device responds with an acknowledgement
  - Address for data involved in the transaction is exchanged
  - Data exchange happens

# Bus Performance

- Measured by bandwidth - the amount of data a bus can transfer for a given length of time
  - Affected by physical design and associated protocols
- A simpler handshake increases bandwidth
- Shorter buses, fewer connected devices, and more data lines increase the speed of the bus and increase the bandwidth
  - More data lines increases the cost of the board
- Delays in transmission due to handshaking, bus traffic, and different clock frequencies

# Bus Width

- Number of data bits a bus can transmit in a given transaction
- To transmit 32 bits of data over a bus with a bus width of 8, then the data needs to be sent in 4 different transmissions

# Resources

T. Noergaard, *Embedded Systems Architecture* Oxford, UK: Elsevier Science & Technology, 2012.

# Appendix L

## WEEK 9 LECTURE SLIDES

# Board I/O

Week 9

# Background

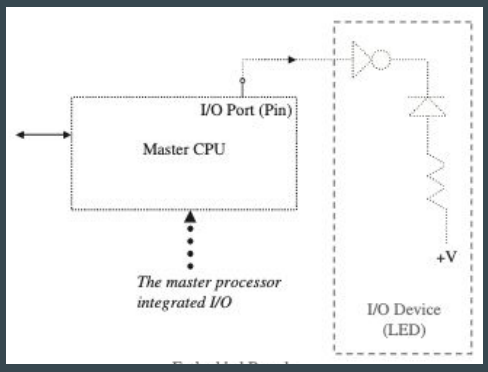
- Output devices receive data from I/O components and display it in some manner
- Input devices transmit data to board I/O components
- Board I/O impacts the system's throughput, execution time, and response time
- Almost any electronic system can be connected to an embedded board and act as an I/O device

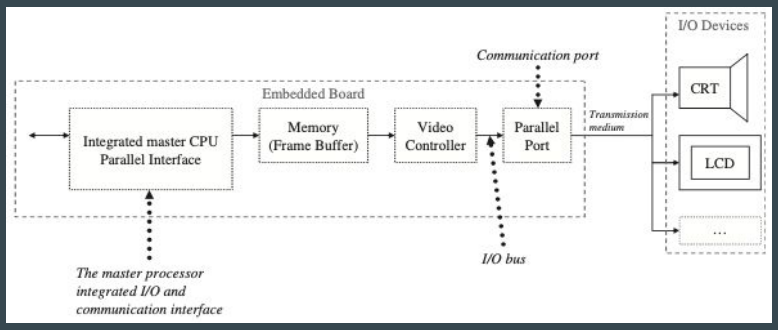
# I/O Hardware

- Transmission medium - wireless or wired medium connecting I/O device to board for data communication
- Communication port - what transmission medium connects to on the board or what receives the wireless signal
- Communication interface - manages data communication between CPU and I/O devices
    - Responsible for encoding/decoding data

# I/O Hardware

- I/O controller - secondary processor that manages I/O device
- I/O bus - the connection between board I/O and main processor
- The main processor integrated I/O



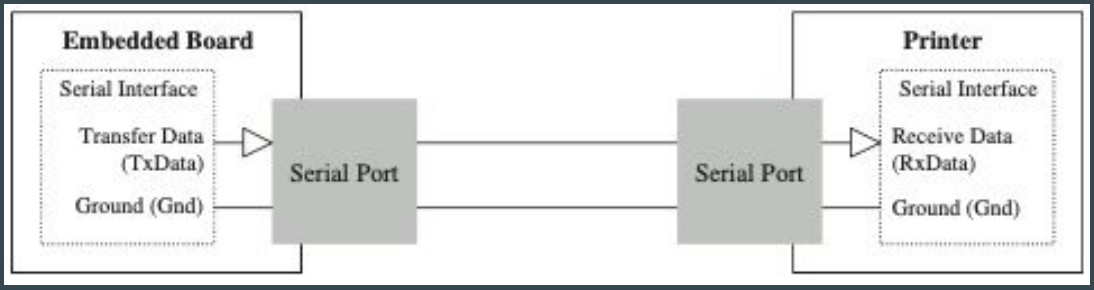



# Serial I/O

- Data is stored, transferred, and received one bit at a time
- Includes a serial port and serial interface
- Serial interface manages data transmission/reception between CPU and I/O device
    - Buffers to store and encode/decode data

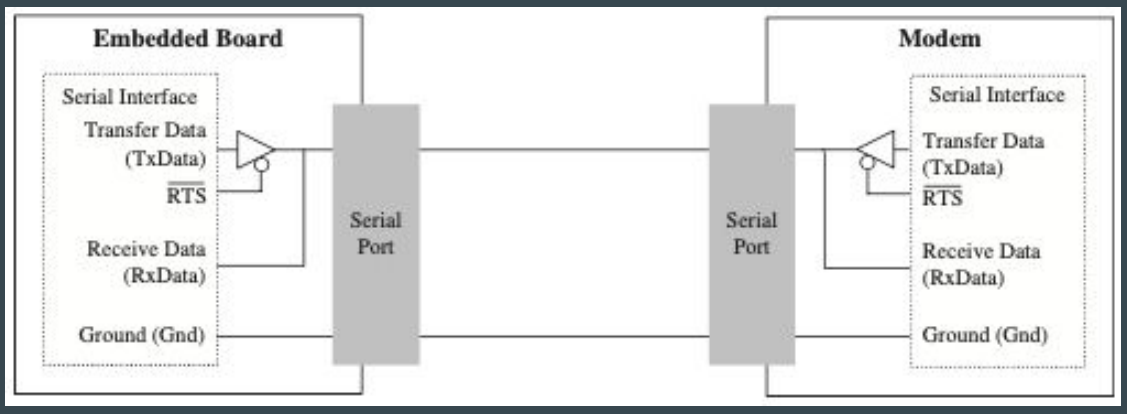
# Simplex Serial I/O

- Data can only be transmitted/received in one direction



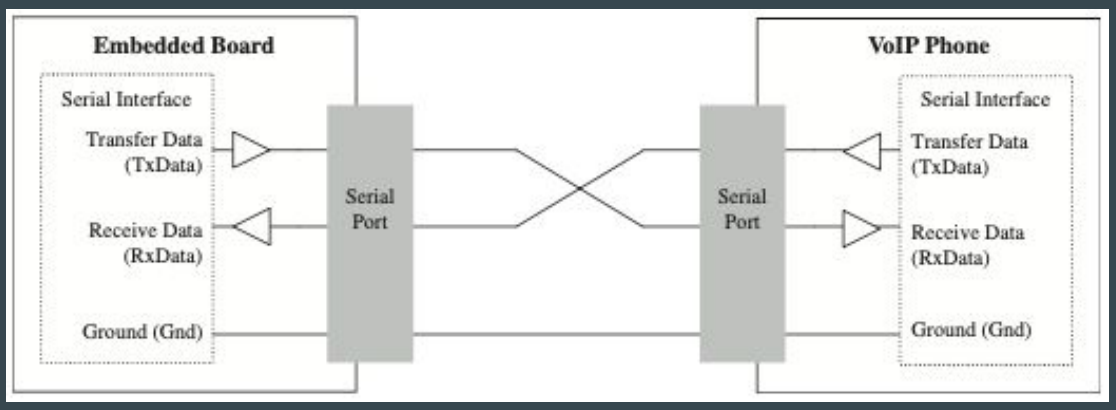
# Half-Duplex Serial I/O

- Data can be transmitted/received in either direction, but only one direction at a time



# Full-Duplex Serial I/O

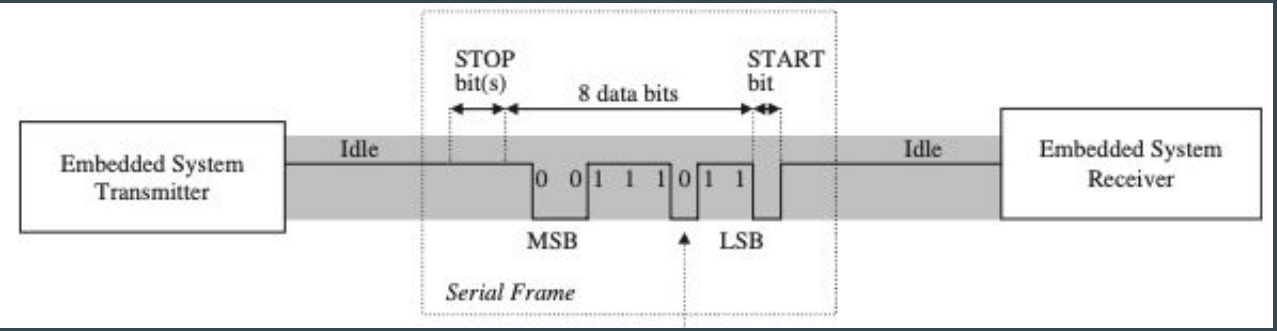
- Data can be transmitted/received in either direction simultaneously



# Asynchronous Serial I/O

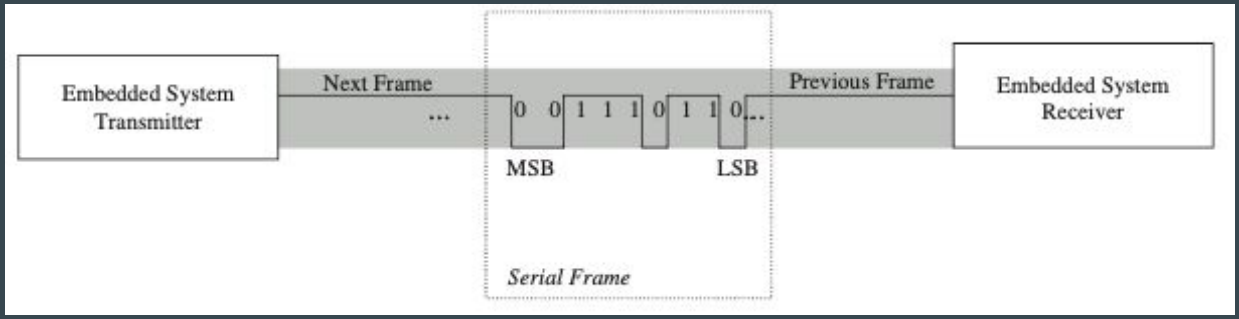
- Intermittent data stream at irregular intervals
- Data is stored and modified in the transmission buffer
- The serial interface creates packets of data
- Packets are then put into frames to be transmitted
  - A frame is a packet with a start bit at the beginning and a stop bit at the end
  - May also include parity bit for error checking
- The communication channel is idle between transmitting frames
- The receiver receives the start bit and starts shifting in bits to receive buffer until it reaches the stop bit
  - The data rate of receiving/transmitting has to be synchronized between serial interfaces
  - Receiving and transmitting devices have their own clock

# Asynchronous Serial I/O



# Synchronous Serial I/O

- Continuous data stream at regular regulated by the CPU clock
- No start and stop bits and no idle period
- Data rates for receiving and transmitting must be in sync
  - Synchronized off of a common clock



# Parallel I/O

- Transfer data in multiple bits simultaneously
- Includes parallel port and parallel interface
- Parallel interface manages data transmission/reception between CPU and I/O device
  - Decodes/encodes data bits
- Can do simplex, half-duplex, and full-duplex transmission
- Can transmit data synchronously and asynchronously

# I/O Performance

- Data rates of I/O devices and the speed of the main processor can vary significantly
  - If the I/O device is much slower than the main processor, then the I/O device may miss data from the processor
  - Need to synchronize speed of I/O device and main processor
- How the I/O device and main processor communicate affects performance
  - Having an I/O controller frees space on the main processor
  - Whether the communication is interrupt-driven, polled, or memory-mapped

# Measure I/O Performance

- Throughput - the maximum amount of bytes per second that can be processed
  - The lowest throughput device drives the performance of the whole system
- Execution time - the time it takes to process all data
- response/delay time - the amount of time between a request to process data and the time the component begins processing

# Resources

T. Noergaard, *Embedded Systems Architecture* Oxford, UK: Elsevier Science & Technology, 2012.