2004-569

Real-Time Operating Systems: A Visual Simulator

Steven F. Barrett[1], Daniel J. Pack[2], Charles Straley[1],
Lew Sircin[1], George Janack[1]

[1]Department of Electrical and Computer Engineering
University of Wyoming

[2]Department of Electrical Engineering
United States Air Force Academy, Colorado

Abstract

A Real-Time Operating System, or RTOS, is an operating environment where multiple events called tasks compete for precious processor operating time of a single processor. The processor must prioritize tasks depending on system requirements to ensure that all tasks complete their required activities. Due to its complex nature, a RTOS is a difficult subject to teach in undergraduate institutions. Often it is difficult for the students to visualize the intricacies and inter-relationships between component parts of the system To help students to 'see' the operations, we have developed a visual hardware simulator that interfaces to an embedded controller. In our application, we use the popular Motorola HCS12 microprocessor as the simulator's host system. The simulator can be easily interfaced with other processor families. The simulator provides a visual display of the status of up to 16 competing tasks. The simulator also provides a keypad for the user to interject new status and a LCD display to illustrate key RTOS activities.

Overview

The subject of Real-Time Operating Systems (RTOS) and their associated concepts are difficult for students to learn. The concepts are quite complex and involve multiple abstract data types such as multiple stacks and linked lists dynamically handing off information to one another in a quickly changing scenario. We developed a visual simulator to help the student view these rapidly changing events. The simulator consists of a hardware board that displays the status of up to 16 different tasks in a RTOS environment. The board was designed to be used with a HCS12-based evaluation board readily available from a number of manufacturers. In this paper we begin with a review of some basic RTOS concepts followed by the inherent complexities involved in effectively teaching these concepts. We then describe in detail the board developed to teach these concepts in a visual environment and suggest scenarios that may be used to illustrate RTOS concepts.

Background

What is a RTOS? A RTOS is a computer operating system hosted on a single processor. Since only a single, sequential processor is employed in such applications, the operating system must respond to multiple events or tasks and ensure that all tasks are given sufficient, precious processing time to complete their required actions [1]. To accommodate multiple tasks, the tasks may be categorized into priorities (high, medium, and low). In general, higher priority tasks should be executed by the processor first; however, the processor must ensure that eventually all tasks are allowed to complete their required actions [2-4].

An effective RTOS system will respond to all tasks competing for the same precious processor time in such a manner that multiple tasks appear to be handled simultaneously. How can a single processor do this? The processor typically uses various scheduling algorithms to allow the highest priority task to execute for some amount of time. The processor will then temporarily suspend the executing task, store details about its current executing environment (its context) [2-5], and then allow a different task to operate for some length of time. An effective scheduling algorithm will appear as if all tasks are operating simultaneously while seamlessly handing off processor execution time from task-to-task.

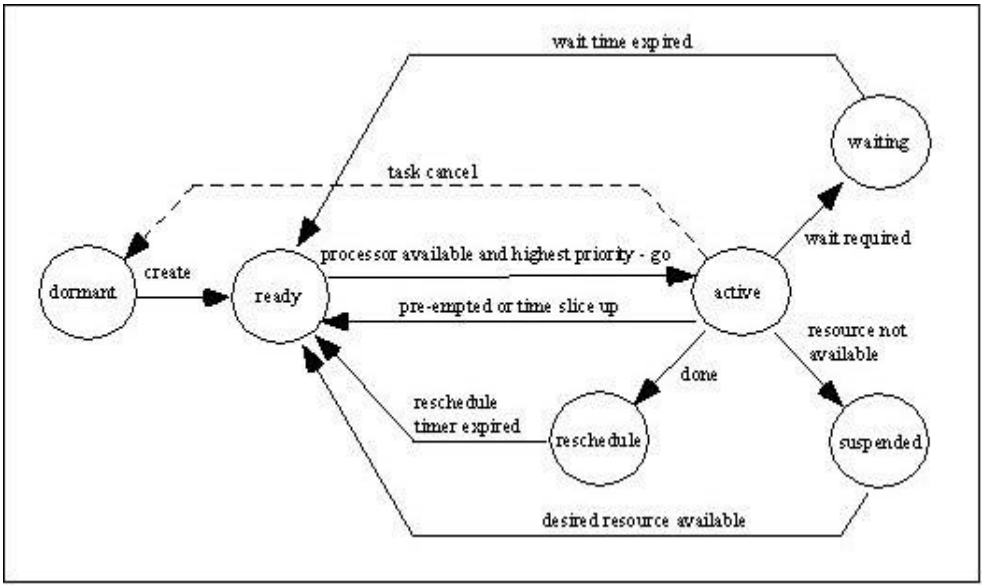

Figure 1. Task state transition diagram [adapted from 8].

To keep track of the progress being made for each task, a set of task states are used. The different possible task states are illustrated in Figure 1. Tasks transition from one state to another in response to external events or operating system updates. A task may be in only a single state at a given time. A task will be in one of the following states [2-5]:

- Dormant (D) - In the dormant state, the task has no need for computer time. It is considered a non-active task. It transitions into the ready state when so directed by the operating system.
- Ready (R) - In the ready state, a task is fully capable of entering the active or running state; however, another task is currently using the processor. A task may enter the ready state from either the dormant state or from the active state.
- Executing/Active/Running (A) - A task in the active state is executing its associated activities on the processor. Since our system only contains a single processor, only one task may be in the active state at any given time.
- Waiting (W) - In the wait state a task has been delayed from execution. It remains in the waiting state for a designated amount of time and then transitions back to the ready state to await processor time. A task is placed in the wait state temporarily to allow lower priority tasks an opportunity to execute.
- Suspended (S) - In the suspended state the task is waiting for some resource. Once the resource is available the task transitions to the ready state and awaits processor time.
- Rescheduling (X) - The rescheduling state is entered whenever a task runs to completion but does not need to be repeated and enter the ready state right away.

A scheduling algorithm is used to determine how tasks are prioritized to obtain processor execution time. A single type of scheduling algorithm does not fit all applications. In fact, there are multiple types of scheduling algorithms briefly described below. A particular scheduling algorithm is employed for a specific application [4-6].

- Polled Loop System – In a polled loop system, the scheduling algorithm sequentially polls various system tasks. A task that is ready is executed until completion. Once task related actions are complete, the scheduling algorithm continues to poll for other ready system tasks. This type of scheduling algorithm is an effective method of completing equal priority tasks that are not likely to occur simultaneously.
- Polled Loop System with priority interrupts – This algorithm is similar to the polled loop system described above. However, a limited number of higher priority tasks are allowed to interrupt the normal sequence of polled tasks. When higher priority tasks become ready to execute, they are allowed to seize control of the processor and execute their tasks related activities until completion. When the interrupt has been properly serviced, the scheduling algorithm returns to its normal polling activities.
- Round-robin Systems – A round-robin system gives each task an equal slice of processor execution time. The scheduling algorithm sequences from one equal priority task to another.
- Hybrid Systems - The round-robin system may also be equipped with a limited number of higher priority interrupt driven tasks. This is referred to as a hybrid system.
- Interrupt Driven Systems - In an interrupt driven system, the processor is placed in a continuous loop. As interrupt tasks become ready, they are executed. If two

tasks become ready at the same time, the task with the highest priority is allowed to execute first.

- Cooperative Multitasking – In this type of scheduling algorithm, a ready task executes for a prescribed amount of time. The task then relinquishes processor control to another ready task.
- Pre-emptive Priority Multitasking System – A pre-emptive priority system is very similar to the Cooperative Multitasking system. However, the scheduling algorithm determines when a task should relinquish control of the processor rather than the task relinquishing control to another task.

How does the processor/algorithm keep track of all this dynamically changing status? A RTOS system uses various abstract data types such as linked lists, records, stacks, and queues to keep track of system status [7]. Linked lists are used to track tasks that are in various states [4]. For example, a separate linked list is used to track which tasks are ready to execute, tasks that are dormant, etc. Reference Figure 2.

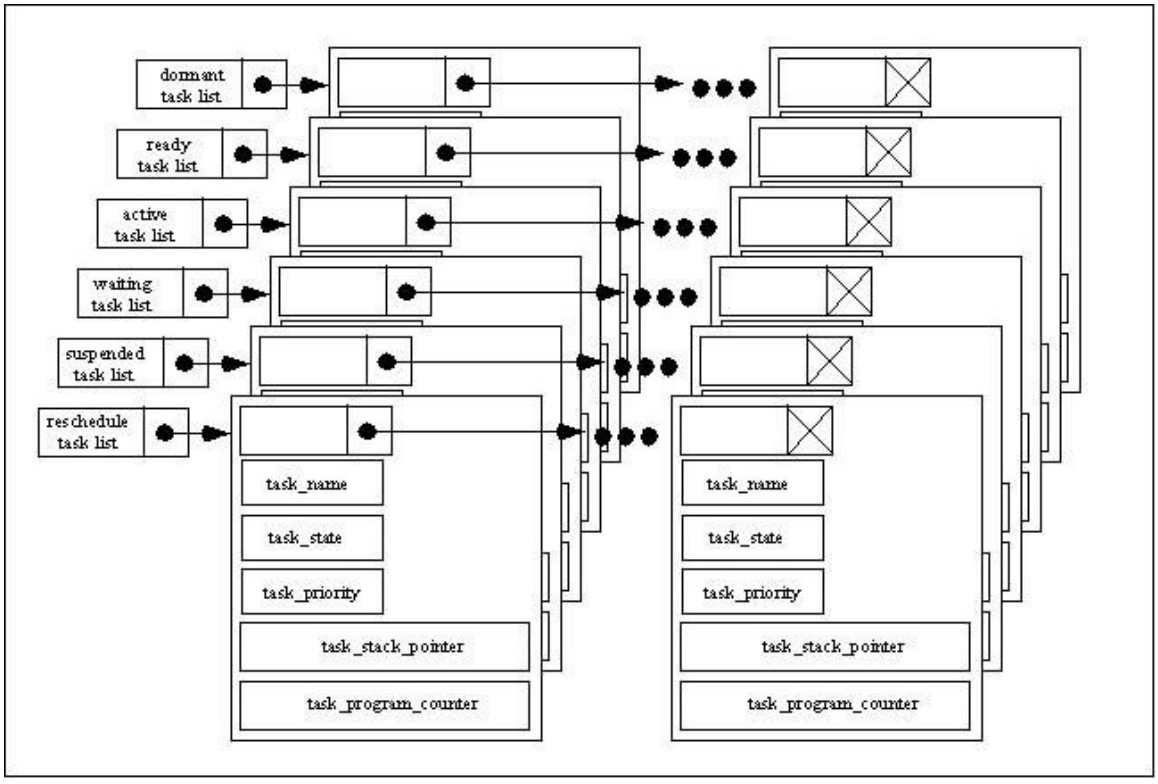

Figure 2. Multiple linked list of tasks. A separate linked list is used to track which tasks are ready to execute, tasks that are dormant, etc. [adapted from 1].

A task is implemented with a record. A record is a collection of different data types providing related information about a given entity [7]. The task record contains the task name, the current task state (ready, dormant, etc.), the task priority, the task stack pointer to keep track of current task status, the task program counter, and a link to the next task in a given task list [5]. The task's context is stored in a stack, another abstract data type.

The stack may be implemented using a fixed array or it can be implemented with another linked list. Other data types such as a circular queue can be used to implement specific scheduling algorithms such as the round-robin scheduling algorithm.

Due to the dynamic nature of abstract data types, a processor with an adequate sized Random Access Memory (RAM) component is required. The RAM is used to implement the heap where the data structures are dynamically allocated and de-allocated during program execution [7, 9].

Why are RTOS concepts difficult to teach? By nature the material is quite complex. Often it is difficult for the students to visualize the intricacies and inter-relationships between component parts of the system Also, the activities are quite dynamic. For example, tasks are constantly changing states and also the multiple linked lists used to track system status are constantly being updated during system operation. For a student to fully understand the concept of RTOS and all of its intricacies, we believe a visual picture of the system is required. To that end, we have developed a visual hardware simulator that interfaces to an embedded controller.

Proposed Solution

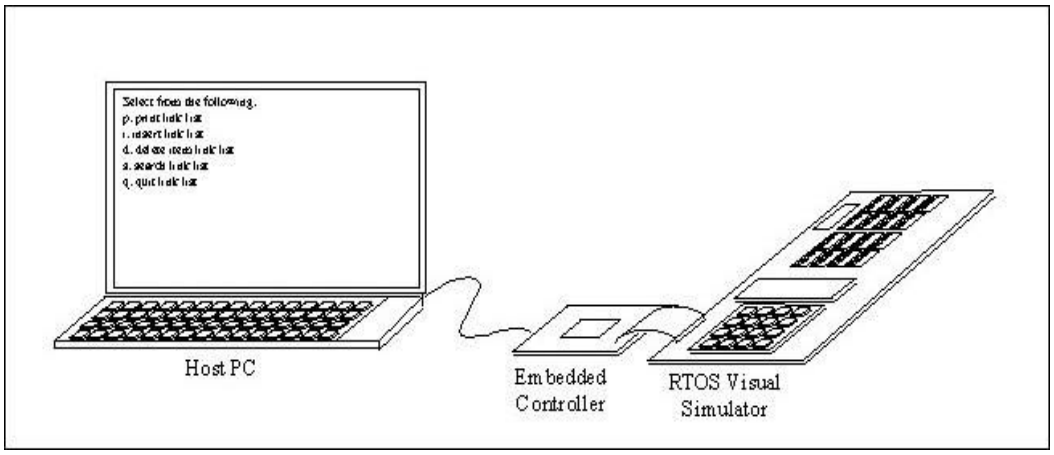

Figure 3. Overview of the RTOS Visual Simulator

As shown in Figure 3, the system consists of a host PC, an embedded controller evaluation board (EVB) and the RTOS Visual Simulator. The host PC contains the system compiler. The compiler is used to program a specific RTOS algorithm. It also hosts the programming pod used to download the RTOS operating system to the target embedded controller. The embedded controller is hosted on an EVB. The RTOS actually executes from the embedded controller. This requires an embedded controller with enough RAM space adequate for the anticipated overhead of multiple, large linked lists with record size elements. The embedded control system also responds to external input and issues output to the RTOS Visual Simulator during RTOS program execution. A logic analyzer may also be used to observe any desired system timing status.

Figure 4 illustrates the RTOS Visual Simulator layout. The RTOS Visual Simulator interfaces to the embedded controller via an edge connector. This allows exchange of data between the embedded controller and the simulator. We have used a Motorola HCS12-based evaluation board with a DP256 processor due to its large RAM component. Also, since this is used in a teaching environment, all code is downloaded and tested in RAM. The RAM is also needed for the heap employed in dynamic memory allocation of the data structures.

The simulator implements sixteen different tasks (0-F). A specific task is activated using a hardware 1:16 decoder (74HC154). The decoder ensures that only a single task is in communication with an embedded controller at a given time. All tasks share a common six-line data bus. The current state (Ready, Dormant, etc.) of a specific task is indicated with an illuminated light emitting diode (LED). The user may interject status via a hexadecimal keypad. The keypad is depressed twice to initiate a specific action. The first switch activation selects the task while the second switch activation controls the activity associated with the task. The task and its associated activity is displayed on the liquid crystal display (LCD).

Figure 5 illustrates the schematic of the task circuit. Each task is implemented with an octal latch (74HC573). A specific latch is enabled using the 74HC154 decoder previously described. This allows task data residing on the task data bus to be latched into the task. The latch will hold the task status until updated. The latch outputs are fed to a LED to indicate the task state. The 7404 hex inverter, 330 ohm series resistor, and the LED form a "logic probe" circuit to indicate the status of the task. The 7404 standard TTL inverter output logic low sink current capability is sufficient for illuminating the LEDs. The full-up hardware system just prior to final fabrication is illustrated in Figure 6.

The Software System

To aid in student understanding of Real Time Operating Systems, a highly volatile, rapidly changing scenario may be used to illustrate RTOS concepts. A scenario may be chosen that is easy to visualize by the students. For example, a used car dealership scenario, a waitron handling multiple customers in a restaurant, or even the security system in a large hotel can be used as an overlay example to illustrate RTOS concepts. Different scenarios may be used to illustrate the different scheduling algorithms.

For example, a used car (task) can have its inherent context such as year, make, model, vehicle identification number (VIN), and odometer reading. The car (task) can be in a variety of states similar to the task states of ready, dormant, etc. In the case of a used car, it may be ready for sale (ready), out for a test drive (active), unavailable due to

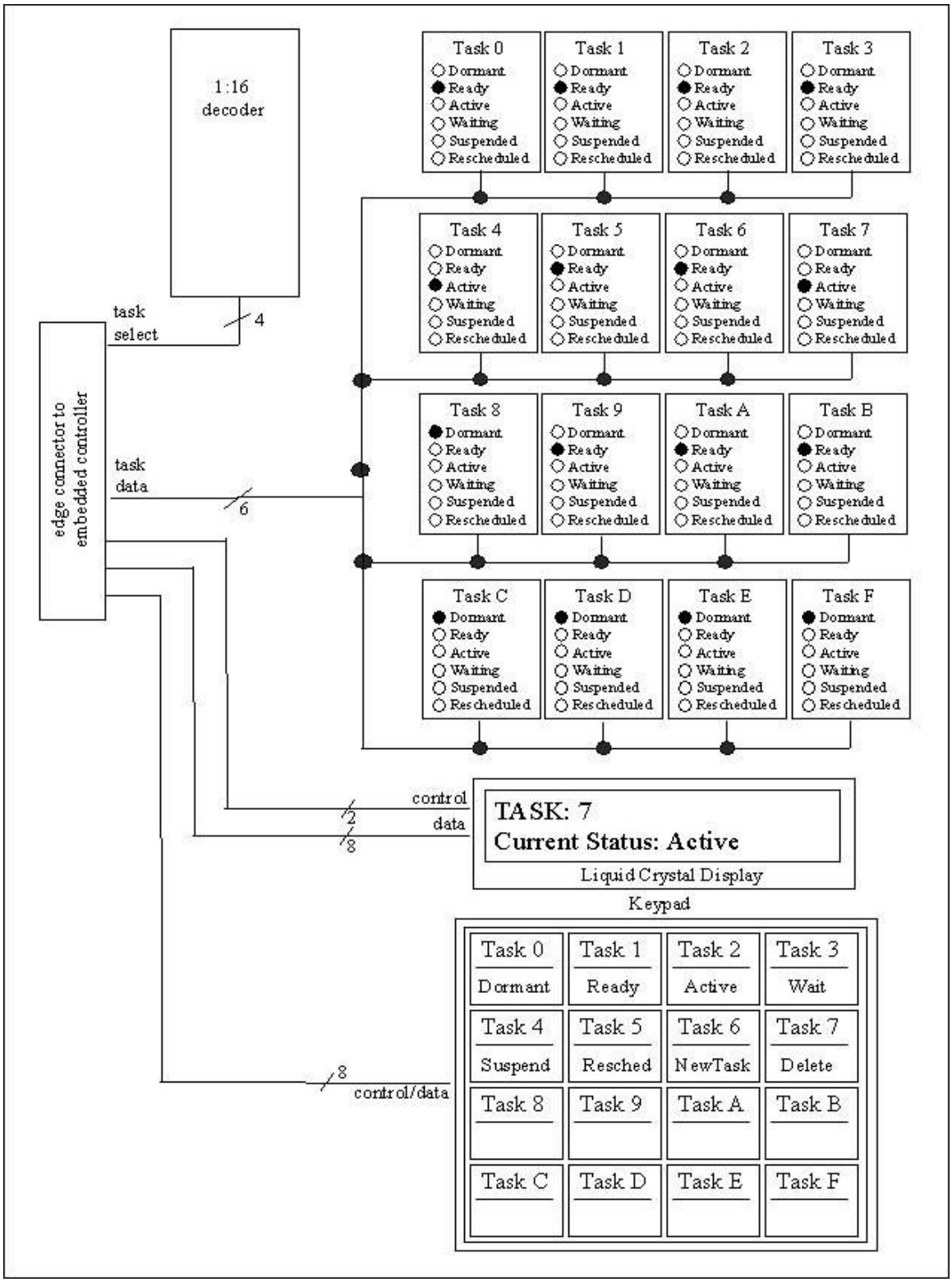

Figure 4. RTOS Visual Simulator Layout

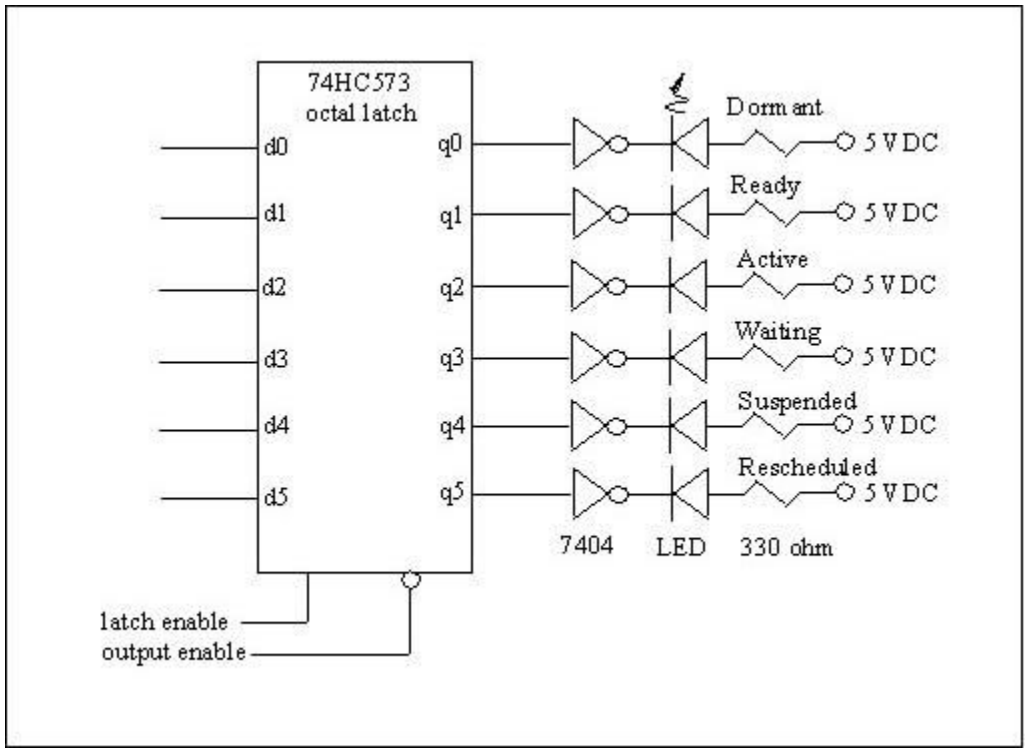

Figure 5. Task Circuit. LEDs illuminate when the corresponding latch output q_n is low.

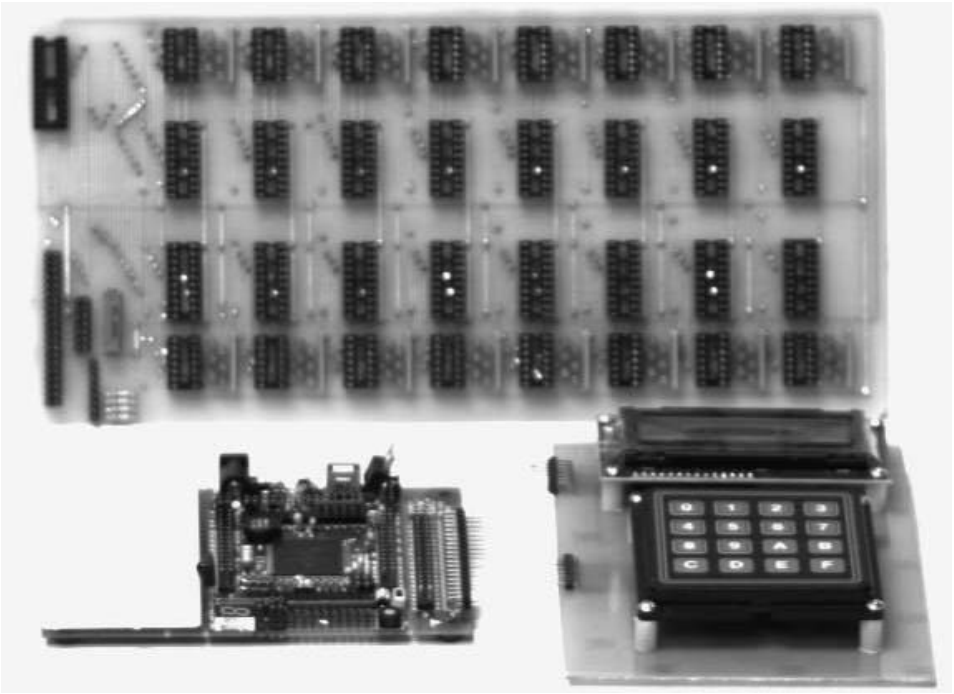

Figure 6. The RTOS Visual Simulator prior to final assembly. The upper PCB contains the "Task Farm" while the PCB on the bottom left is a Minidragon HCS12 DP256 based evaluation board (Wytec, Inc.). The PCB on the lower right hosts the hexadecimal keypad and LCD. (Task Farm and keypad PCBs were designed and fabricated by Lew Sircin and George Janack.)

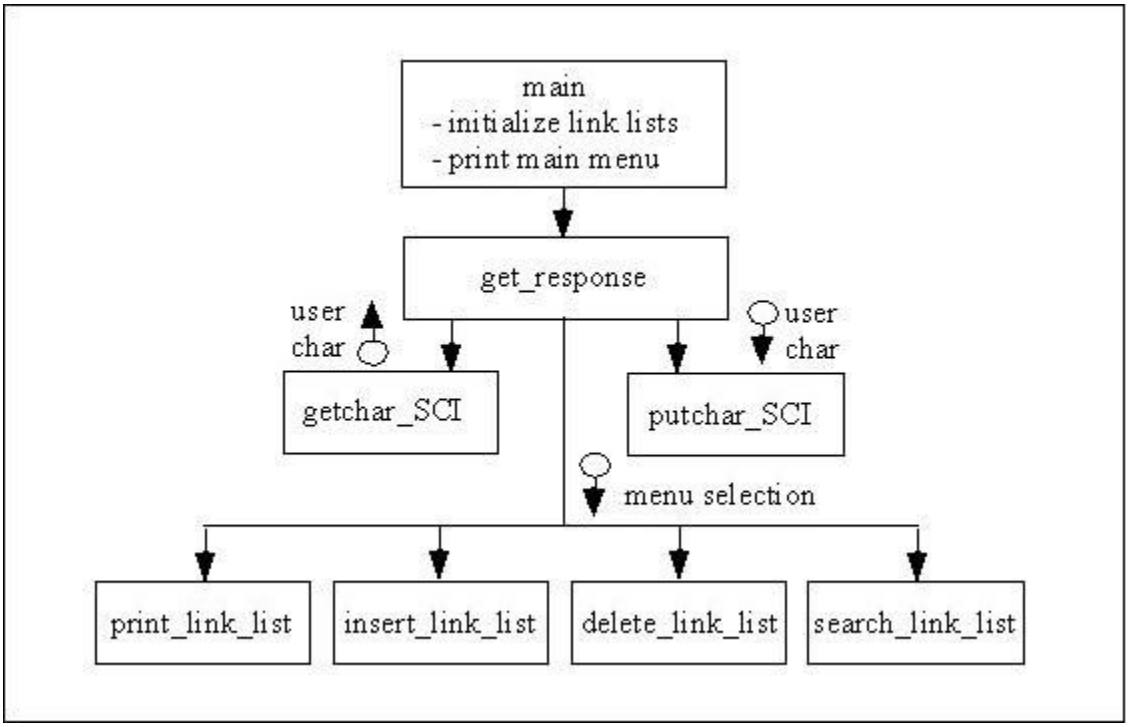

Figure 7. Structure Chart of Car Lot RTOS System.

maintenance (waiting), dormant (sold), etc. The status of up to sixteen cars (tasks) may be displayed on the RTOS Visual Simulator. New scenario status may be injected by an user with the hexadecimal keypad. The new status may be shown on the LCD and the contents of the linked lists in response to the status changes may be displayed on the PC screen.

The structure chart for the Car Lot RTOS System is provided in Figure 7. It provides the basic features required to operate the car lot scenario. The basic software system consists of functions to initialize the system and to perform basic linked list processing such as print a linked list, insert a new item to the linked list, delete a specified item from the linked list, and search for a specific item in a linked list. This code could be provided to the students to become familiar with basic linked list operations. As a homework or laboratory assignment, students could add the necessary software for the following activities:

- The control and data signals to activate status changes on the RTOS Visual Simulator,
- The software interface for the hexadecimal keypad,
- The software interface for the Liquid Crystal Display, and
- Additional scenario features.

Summary/Conclusions

Real Time Operating Systems (RTOS) and their associated concepts are difficult to teach. The concepts are quite complex and involve multiple abstract data types such as multiple stacks and linked lists dynamically handing off information to one another in a quickly changing scenario. We have developed a visual simulator to help the student view these rapidly changing events. The simulator consists of a hardware board that displays the status of up to sixteen different tasks in an RTOS environment. The board was designed to be used with a HCS12-based evaluation board readily available from a number of manufacturers.

The system may be used with a variety of overlay scenarios so the student may become familiar with the complex operation of a RTOS. Once students are comfortable with the basic RTOS concepts, the simulator may be used to evolve into a stand alone RTOS system. Other complex RTOS attributes such as interacting tasks, sharing resources, intertask communications, concurrency, and re-entrancy issues may also be added [5]. If you are interested in any of the items discussed in this paper, please contact us at steveb@uwyo.edu.

References

1. M. Podanoffsky, "Building a Real-Time Multitasking Executive," Circuit Cellar Ink, The Computer Applications Journal, No. 27, June/July 1992, pp. 14-21.
2. J. Ganssle, "Writing a Real-Time Operating System - Part I A Multitasking Event Scheduler for the HD64180," Jan/Feb 1989, Circuit Cellar Ink, pp. 45-51.
3. J. Ganssle, "Writing a Real-Time Operating System - Part II Memory Management and Applications for the HD64180," Mar/Apr 1989, Circuit Cellar Ink, pp. 30-33.
4. P. Laplante, "Real-Time Systems Design and Analysis An Engineer's Handbook," IEEE Computer Society Press, 1993.
5. G.H. Miller, "Microcomputer Engineering," second edition, Pearson Education, 1998.
6. J. Ganssle, "An OS in a CAN," Embedded Systems Programming, Jan 1994.
7. J.F. Korsch and L.J. Garrett, "Data Structures, Algorithms, and Program Style Using C," PWS-Kent Publishing Company, 1988.
8. S.F. Barrett and D.J. Pack, "68HC12 Microcontroller: Embedded Systems Design and Applications," Prentice-Hall Inc, 2004.
9. "ICC12, ImageCraft C Compiler and Development Environment for Motorola HC12," Image Craft Creations, Inc., 2001.

Steven F. Barrett received the BS Electronic Engineering Technology from the University of Nebraska at Omaha in 1979, the M.E.E.E. from the University of Idaho at Moscow in 1986, and the Ph.D. from The University of Texas at Austin in 1993. He was formally with the United States Air Force Academy, Colorado and is now an Assistant Professor of Electrical and Computer Engineering, University of Wyoming. He is a member of IEEE (senior), Tau Beta Pi (faculty advisor), and serves as the President, Rocky Mountain Bioengineering Symposium, Inc. His research interests include digital and analog image processing, computer-assisted laser surgery, and embedded controller systems. He is a registered Professional Engineer in Wyoming and Colorado. He co-wrote with Dr. Daniel Pack "68HC12 Microprocessor: Theory and Application," Prentice-Hall, 2002 and "Embedded Systems Design and Applications with the 68HC12 and HS12," Prentice-Hall, 2004.

Daniel J. Pack is a Professor in the Department of Electrical Engineering at the United States Air Force Academy, CO. He received the Bachelor of Science degree in Electrical Engineering in 1988, the Master of Science degree in Engineering Sciences in 1990, and the Ph.D. degree in Electrical Engineering in 1995 from Arizona State University, Harvard University, and Purdue University, respectively. He was a visiting scholar at Massachusetts Institute of Technology-Lincoln Laboratory. He is a member of Eta Kappa Nu, Tau Beta Pi (faculty advisor), IEEE (senior), and ASEE. He is a registered Professional Engineer in Colorado. His research interests include intelligent control, automatic target recognition, and robotics. Email: daniel.pack@usafa.edu

Charles Straley received the AS Engineering from the Western Wyoming Community College in 2000, and the BS Electrical Engineering from the University of Wyoming in 2004. He is a member of IEEE, ISA, Tau Beta Pi, and Phi Theta Kappa.