Laboratory for Real-Time and Embedded Systems

Milan E. Soklic, Ph.D.

Software & Electrical Engineering Department
Monmouth University, West Long Branch, NJ
msoklic@monmouth.edu

Abstract

This article discusses the design and implementation of laboratory equipment suitable for teaching and research in the area of embedded and real-time systems.

Basic characteristics of real-time systems are that they are embedded and inherently concurrent. Being embedded implies that interfaces of software modules are to be well defined, whereas, concurrency indicates that software processes execute logically in parallel. Using a uniprocessor computer, software processes execute in an overlapped fashion and ordering of events and communication among the processes is not always what was expected. To bridge the gap between the theory and the practice, a laboratory was built which supports the study of embedded and real-time issues that appear simple on the surface, but are in reality “wicked” problems.

The real-time laboratory equipments is designed and assembled around several commercially available components. Its main software/hardware components are personal computer workstations hosting Tornado integrated tool for development of real-time and embedded software, single board computers hosting VxWorks real-time operating system, and a toy-size hardware railroad.

The described laboratory equipment enables the user to have an ample opportunity to gain an in-depth understanding of the underlying mechanisms that govern the behavior of embedded and concurrent processes, to compare software execution in simulated and in real, i.e. native, environment, and to visually observe and verify software behavior on the real-time controlled demonstration target – the railroad model.

The real-time laboratory started with the author's grant obtained from the WindRiver company a manufacturer of Tornado integrated software development environment for embedded an real-time systems.

Real-time and embedded systems

Systems are called embedded if they are a part of larger systems. Embedded systems range in size and complexity. They can be found in telephones, toys, and in more complex systems, such as, automobile cruise control systems, process control systems, systems for medical diagnosis, telecommunication systems, and defense systems. A personal computer, for instance, can be seen as having many embedded systems, which represent specialized I/O cards. Embedded systems might have specific design constraints, such as, small memory size, fail-safe operation, low power consumption, high reliability, and high safety.

Systems that provide correct results in a timely manner are called real-time systems. They are, by and large, designed as a set of concurrent and cooperating processes. Computations are divided into tasks that communicate either by synchronous message passing, or by asynchronous message passing. Real-time embedded applications are predominantly concerned with starting and completing the tasks at the most valuable times, neither too early nor too late. In short, the real-time systems must satisfy scheduling constraints while processing their data.

Traditionally, real-time systems are built by first developing application software, and then, validating timing constraints and the suitability of the programming language used. Fine-tuning of timing constraint is in many cases accomplished by extensive simulation. This approach is at least time consuming, and maintenance of the resultant software system is difficult, since small changes in application software can produce unpredictable results.

The intellectual gap which exists between the theory, i.e. a set of assumptions about how to design and implement software for real-time systems, and the way software solutions are actually carried out on a physical computer, can be investigated using appropriate equipment. In general, two broad issues need to be considered: On one hand, the performance of a real-time computer controlled system depends heavily on its memory subsystem, the microprocessor technology, and the interfacing subsystem that provides coupling of the computer system to its target environment. Consequently, the microprocessor pipelining, the context switch time, and the memory hierarchy subsystem made the prognosis of 'exact' timing of software execution difficult, if not impossible. On the other hand, models for real-time software controlled systems do not explicitly make use of time, and are, as a result, bound by the solutions provided by scheduling algorithms and approximate deadlines of starting and terminating the software processes involved. However, the environment to be controlled is typically time variant and continuous, whereas, the software processes, which interact with this environment, are by default designed to terminate, and have no notion of time. To bridge the gap between the theory and practice, to gain a deeper understanding of issues related to real-time architectures, real-time programming, and real-time scheduling strategies, a suitable equipment is needed.

The laboratory equipment

To address the issues illustrated above, laboratory equipment was built which supports the study of real-time and embedded systems.

The laboratory equipment is designed and assembled around several commercially available software and hardware components. The bock diagram in Figure 1 depicts major building components and their connections into laboratory equipment. The major hardware/software components are: a personal computer workstation (PCW) hosting Tornado integrated tool for development of real-time and embedded software, a single board computer (SBC) hosting VxWorks real-time operating system, and a model of a hardware railroad.

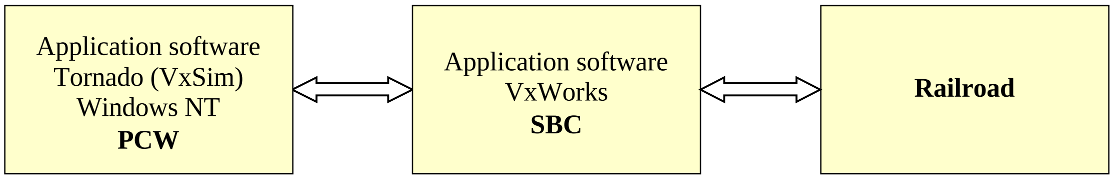

Figure 1: The configuration of subsystems into equipment for development of real-time and embedded software.

Application software is developed, debugged, and tested in the Tornado environment. Tornado executes on Windows NT hosted by the personal computer workstation. The real-time operating system VxWorks, hosted on the single board computer, provides a native environment for application software. The railroad is pure hardware with no notion of a software program.

The laboratory can be build incrementally, starting with Tornado, adding the single board computer, and finally the railroad. This approach of building a system, in turn, increases system capability in a step-by-step manner, and at the same time allows the user to use the partial system before it is complete. Thus, for example, having only Tornado, application software can be developed and executed in a simulated environment VxSim, provided by Tornado. Next, the connection of single board computer to the personal computer workstation enables application software developed in Tornado environment to be downloaded to the single board computer and executed in the native environment of VxWorks. Finally, by connecting the railroad, application software for the railroad control can be downloaded to the single board computer. Application software controls the railroad via an application-programming interface which utilizes the railroad hardware components. The application-programming interface is a point that bridges the gap between software and hardware, that is, software instructions are converted into a form directly interpretable by the underlying railroad hardware.

The following sections describe the individual subsystems in more detail, and outline some possible studies around the real-time software controlled railroad model.

Tornado and VxWorks highlights

Tornado provides an integrated software environment for developing, debugging, testing and tracing the execution of embedded and real-time software. The environment supports software development using C/C++ programming language and function calls to the VxWorks real-time operating system. VxWorks acts as a real-time executive that provides support for concurrency, interprocess communication, and management of resources. A virtual memory manager can be added at the expense of a slower system [1].

Concurrency is accomplished by means of CPU (or task) scheduling mechanisms that considers task priorities. A priority is an indication of a relative importance of a task. Task priorities can be assigned in different ways whose outcome can result in different scheduling strategies, such as: round robin scheduling where all tasks have equal priority and the same time slice, rate monotonic scheduling where a task with the shortest, i.e. projected, CPU request time is given a higher priority, and shortest deadline first scheduling where the task with the most immediate need gets the highest priority. To address some of the scheduling strategies, VxWorks provides a scheduling mechanism illustrated in Figure 2.

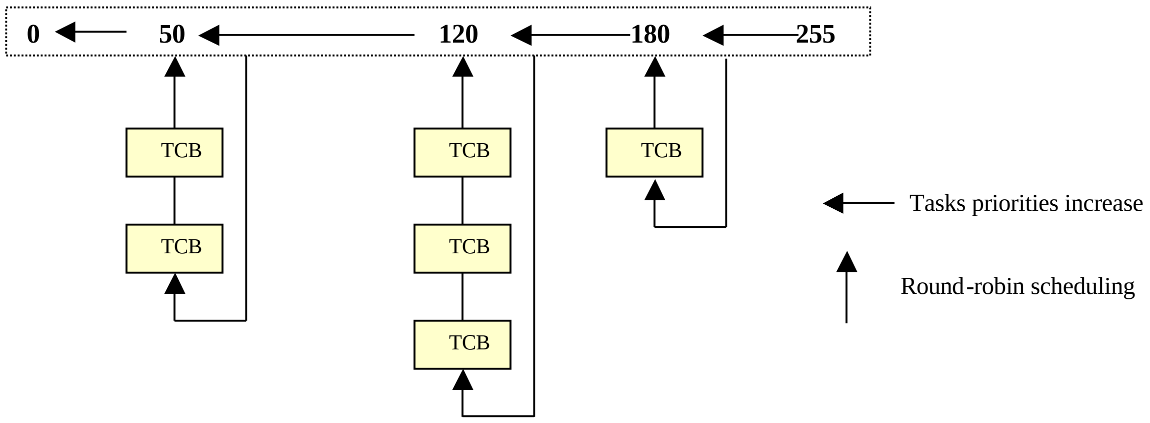

Figure 2: Organization of VxWorks scheduling mechanism.

VxWorks allows 256 priorities, with the highest priority numbered zero, and the lowest priority numbered 255. The scheduling mechanism is implemented as a multiple double-linked list of task control buffers (TCB). A single double-linked list belongs to a unique priority number. Thus, tasks with the same priority, shown in vertical lines on Figure 2, are executed in a round-robin fashion, i.e. first-in first-out queue with the same time slice. The time slice is global.

Tasks with different priorities are in a horizontal line. A task with a higher priority immediately pre-empts a task with a lover priority. Figure 2 shows an example of six tasks: two tasks with priority of 50 scheduled in round-robin fashion, three tasks with priority of 120 also using round-robin scheduling algorithm, and one task with priority of 180.

Application software, after cross-compiled, is executed using VxSim, a prototyping and simulation tool for VxWorks environment. The execution involves dynamic linking and loading of software programs. To trace a detailed execution of application software the WinView tool, considered as a software logical analyzer, can be used to graphically display a history of software execution. To get the smallest possible memory footprint for the target hardware, Tornado enables customizing VxWorks real-time kernel to the application demands. Tornado comes with abundance of manuals, many of which are also available online.

The role of the single board computer is two fold: first, to execute application software in the native operating environment of VxWorks, and second, to use the board as a real-time computer control of the railroad model. For the above purposes the Motorola single board computer PrPMC800 is used[2]. The board utilizes MPC7410 microprocessor, based on the RISC technology, which enables a shorter context switch time compared to the CISC technology.

Railroad in more detail

To gain hands-on experience in real-time control of a man-made environment, a reasonably complex model of a railroad system was designed, constructed, and populated with several trains. The block diagram in Figure 3 shows conceptual organization of the major hardware modules into a into a digital railroad system manufactured by the Marklin Company[3].

The control unit is a gateway for sending commands to the middle rail of the track; illustrated with a dotted line on Figure 3. The commands, coming from the single board computer (SBC), include control of track-switches and control of locomotives. The track detection module gathers data from sensors mounted along the track. The data from the track detection module are sent to the single board computer upon request of application software. The interface accommodates signals between the single board computer and the railroad.

The track consists of numerous track parts, of various lengths and shapes, connected together to form a track layout. The layout can accommodate several trains, i.e. locomotives and cars. The track layout represents all possible trajectories a train makes during its movement. The actual trajectory made by a train, when moving along the track, depends on the position of the track-switches scattered along the track and the direction of the train movement. The trajectory of a train can be periodic or aperiodic. If aperiodic, the train does not traverse through the same sections of the track, due to the direction of train moment and/or due to possible changes of the track-switches. Briefly, changing the settings of the track-switches makes a train trajectory dynamic.

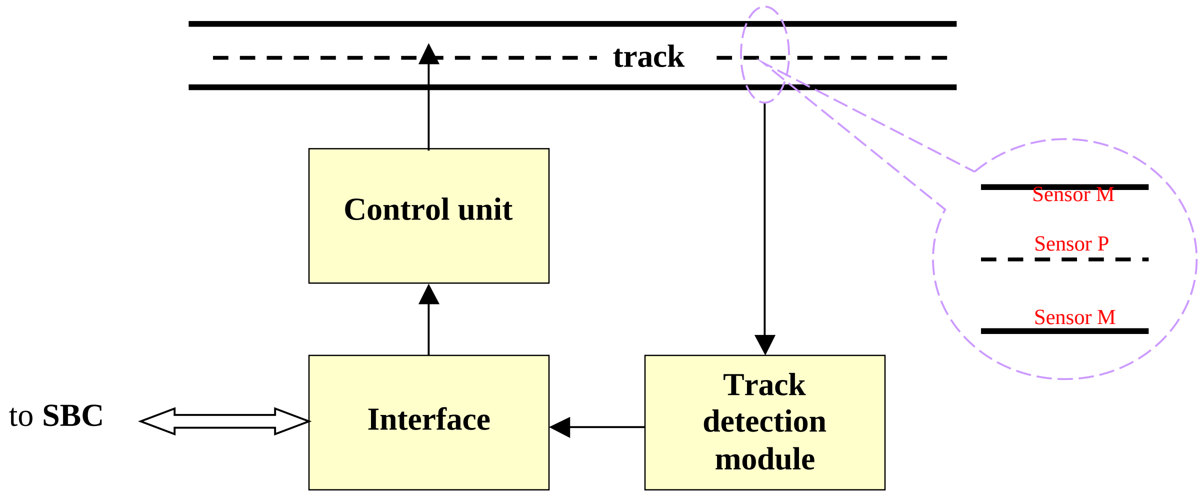

Figure 3: Principal arrangement of modules into a digital railroad system.

To detect the presence or absence of a moving train sensor-points positioned along the track are used. A single sensor-point, marked with a dashed oval line across the track on Figure 3, can have three sensors. A sensor has a built-in contact switch that short-circuits its terminals, if a force is applied to that sensor.

Two type of sensors are used in the layout: the M-type sensor, triggered by the magnetic force emanating from a magnet attached to a train, and the P-type sensor, triggered by a physical force of a moving locomotive pick-up shoe. A part of the Figure 3 shows the enlarged portion of a sensor-point with three sensors. The sensors can only detect moving trains in their close vicinity, and cannot identify them. The P-type sensor can also detect the direction of a train movement. Each sensor is electrically connected to a latch in the track detection module. When a force triggers a sensor, the build-in switch makes a momentary closure on its terminals, which sets its corresponding latch to logic high, indicating a train was in the vicinity of this sensor.

Application software, running on the single board computer (SBC) controls the railroad via the application-programming interface, a collection of software procedures used to convey control (signals) to the railroad and to receive data from the railroad. Table 1 summarizes thirteen possible software generated controls for the railroad. Thus, a selected locomotive can be controlled by nine different software controls. Several controls can be applied concurrently, for instance, a forward moving locomotive with accelerating speed, blinking front lights, and whistle on every five seconds for a duration of one second.

To control individual devices, i.e. track-switches and locomotives, all the devices have built-in decoders to recognize their messages. A message format consists of the device address, followed by the function to be performed.

Table 1: Software controls of the railroad.

Locomotive:	Stop, Start, Move forward, Move backward, Set speed, Front lights on/off, Back lights on/off, Whistle on/off, Telex coupling on/off
Track-switch:	Straight, Branch
Track detection module:	Reset, Read

Application software gathers sensors' data from the track detection module by means of two software controls, shown in the last row of Table 1. After a sensor is triggered, a latch of the corresponding sensor remains at logic high as long as application software does not read or reset this latch. Thus, to have current sensor values, they should be read frequently enough, to avoid multiple triggering of sensors between their readings.

The railroad is a demonstration target for the real-time software control of an engineering environment. The complexity of controlling the railroad depends mainly on the number of trains used and on the dynamicity of the track layout, i.e. the frequency at which the layout changes trains' effective trajectories. A single train with a static track layout, i.e. with a fixed train trajectory, requires the lowest level of control complexity.

In order for application software to "see" trains on the track layout a feedback from the railroad is needed. For that purpose, application software periodically polls the track detection module, which captures information about track-sensors. Utilizing feedback from the railroad makes it possible to exercise additional objectives, such as: to control the positioning of a train on the track, to determine the speed and length of a moving train, and to activate these controls when a train in question reaches a specified sensor-point. In addition, the identification of a train address is possible if one or more magnets are attached, in a different pattern positions, along the train. The attached magnets trigger the M-type sensors.

The complexity of the railroad increases with the increasing number of locomotives moving with different speeds on a fixed track layout. In such an arrangement, the main purpose of controlling the railroad is to avoid train collisions, while the trains have to carry out their required behaviors. Finally, further increase in the complexity of the railroad is accomplished if the dynamic changes of trains' trajectories are allowed.

Several design approaches can be used to reach at solutions to problems exposed by the described railroad. In general, application software for real-time controlled railroad has to accomplish two different kinds of tasks: hard real-time, and soft real-time. The tasks that are responsible for collision avoidance are hard real-time, since failure to perform this function means fatal failure of the system. Avoiding collision requires the design of algorithms that detect the moving trains on time and act accordingly, either by increasing or decreasing particular trains' speeds, changing appropriate track-switches, or both. A task is considered to be soft real-time, such as, turning the lights on/off, if failure to perform this function in a timely manner temporarily degrades system less significant behaviour.

Last but not least, the amalgamation of several locomotives moving at different speeds and in different possible directions on the track layout, which can dynamically change trains' trajectories causes the emergence of RR holistic features, i.e. features that emerge as a phenomenon of the whole, and are due to the system complexity and interaction of its components.

Conclusions

This article starts by addressing a need for laboratory equipment, pointing out some of embedded and real-time issues, and continues with the description of an overall configuration of main laboratory subsystems, which are then described in more detail, and concludes by suggesting some strategies suitable for utilizing the complexity of the described equipment.

Since there is no substitution for hands-on experience, laboratory equipment was designed and implemented to address issues related to real-time and embedded systems, such as: architectural models for software design of real-time systems, pros and cons of a programming language used, coordination of concurrent tasks using scheduling algorithms, and software interfacing for a computer controlled environment.

There is no programming language that is a good fit for all problem domains. Support for concurrency and management of resources, which are absent in C/C++ programming language, are provided by the real-time VxWorks operating system. This approach makes programming less portable since the language must call for the support provided outside the language run-time system; however, programming in C/C++ makes programs very efficient. A strong typing and good exception handling are very desirable features when programming embedded and real-time systems. Understanding programming language features is essential to avoid programming constructs and techniques that are inherently error-prone. Defensive programming is to be used which involves incorporating checks for faults and fault recovery code in application software.

Models for real-time architectural design lack presence of time. The underlying Boolean logic of digital computers makes digital computation timeless, opposed to analog and hybrid computations, which are time reliant. The absence of time in the design of real-time systems, which must consider time as a part of its correctness is compromised by solutions, such as, scheduling policies and deadlines for tasks completion. Task priorities and their timing constraints provide a crude tool and do not capture the requirements of completion, or initiation, at the most valuable time. As a consequence, real-time application software needs fine-tuning.

Many real-time and embedded systems interact with their physical environment using a variety of sensors and actuators. If software has to interact with a physical environment, it must, as a result, mimic some properties of this environment. There is apparent gap between digital computation, which has no concept of time, and the interacting physical environment, whose processes, perceived as stimuli to the computer-controlled system, are time variants. The issues outlined here can be studied using the described laboratory equipment.

Bibliographical information

[1] Tornado (www.wrs.com)
[2] Motorola (www.motorola.com)
[3] Marklin (www.marklin.com)

Biographical information

MILAN E. SOKLIC is a professor of software and electrical engineering at the Monmouth University in New Jersey. His research interests include software engineering, particularly theoretical and practical aspects of real-time and embedded systems. He is a member of the ASEE.