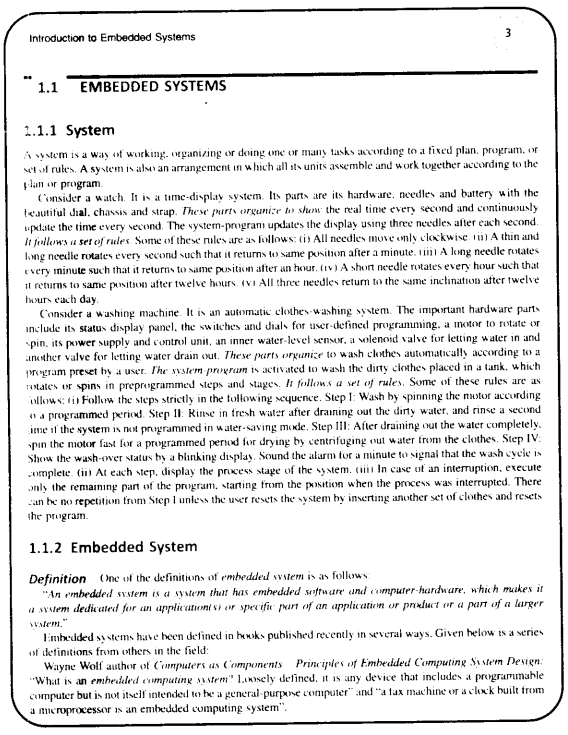
Walkthrough



1.1 EMBEDDED SYSTEMS

1.1.1 System

A system is a way of working, organizing or doing one or many tasks according to a fixed plan, program, or set of rules. A system is also an arrangement in which all its units assemble and work together according to the plan or program.

Consider a watch. It is a time-display system. Its parts are its hardware, needles and battery with the beautiful dial, chassis and strap. *These parts organize to show* the real time every second and continuously update the time every second. The system-program updates the display using three needles after each second. *It follows a set of rules.* Some of these rules are as follows: (i) All needles move only clockwise. (ii) A thin and long needle rotates every second such that it returns to same position after a minute. (iii) A long needle rotates every minute such that it returns to same position after an hour. (iv) A short needle rotates every hour such that it returns to same position after twelve hours. (v) All three needles return to the same inclination after twelve hours each day.

Consider a washing machine. It is an automatic clothes-washing system. The important hardware parts include its status display panel, the switches and dials for user-defined programming, a motor to rotate or spin, its power supply and control unit, an inner water-level sensor, a solenoid valve for letting water in and another valve for letting water drain out. *These parts organize* to wash clothes automatically according to a program preset by a user. *The system-program* is activated to wash the dirty clothes placed in a tank, which rotates or spins in preprogrammed steps and stages. *It follows a set of rules.* Some of these rules are as follows: (i) Follow the steps strictly in the following sequence. Step I: Wash by spinning the motor according to a programmed period. Step II: Rinse in fresh water after draining out the dirty water, and rinse a second time if the system is not programmed in water-saving mode. Step III: After draining out the water completely, spin the motor fast for a programmed period for drying by centrifuging out water from the clothes. Step IV: Show the wash-over status by a blinking display. Sound the alarm for a minute to signal that the wash cycle is complete. (ii) At each step, display the process stage of the system. (iii) In case of an interruption, execute only the remaining part of the program, starting from the position when the process was interrupted. There can be no repetition from Step I unless the user resets the system by inserting another set of clothes and resets the program.

1.1.2 Embedded System

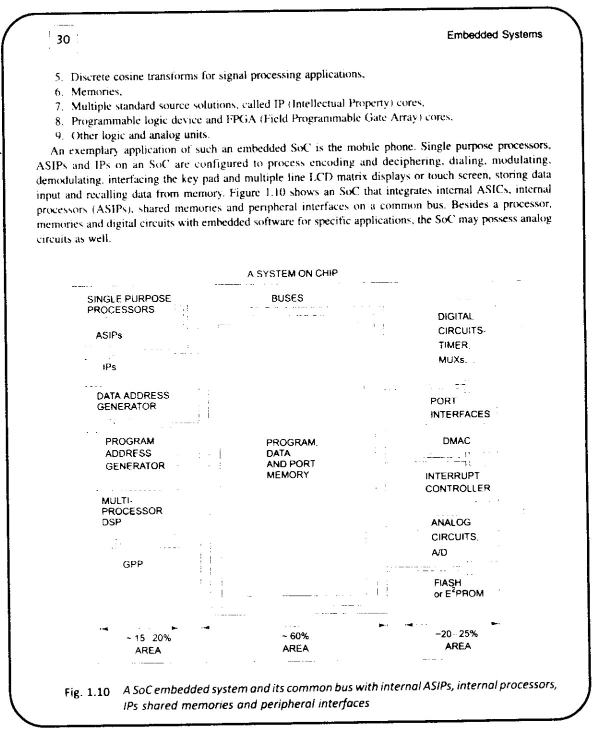
Definition One of the definitions of *embedded system* is as follows:

"An embedded system is a system that has embedded software and computer-hardware, which makes it a system dedicated for an application(s) or specific part of an application or product or a part of a larger system."

Embedded systems have been defined in books published recently in several ways. Given below is a series of definitions from others in the field:

Wayne Wolf author of *Computers as Components Principles of Embedded Computing System Design*: "What is an *embedded computing system*? Loosely defined, it is any device that includes a programmable computer but is not itself intended to be a general-purpose computer" and "a fax machine or a clock built from a microprocessor is an embedded computing system".

Simple approach with interesting examples and figures



5. Discrete cosine transforms for signal processing applications,
6. Memories,
7. Multiple standard source solutions, called IP (Intellectual Property) cores,
8. Programmable logic device and FPGA (Field Programmable Gate Array) cores.
9. Other logic and analog units.

An exemplary application of such an embedded SoC is the mobile phone. Single purpose processors, ASIPs and IPs on an SoC are configured to process encoding and deciphering, dialing, modulating, demodulating, interfacing the key pad and multiple line LCD matrix displays or touch screen, storing data input and recalling data from memory. Figure 1.10 shows an SoC that integrates internal ASICs, internal processors (ASIPs), shared memories and peripheral interfaces on a common bus. Besides a processor, memories and digital circuits with embedded software for specific applications, the SoC may possess analog circuits as well.

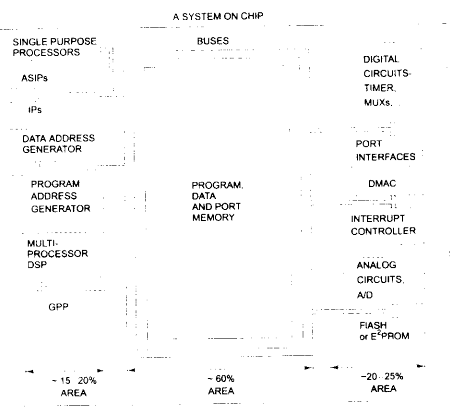
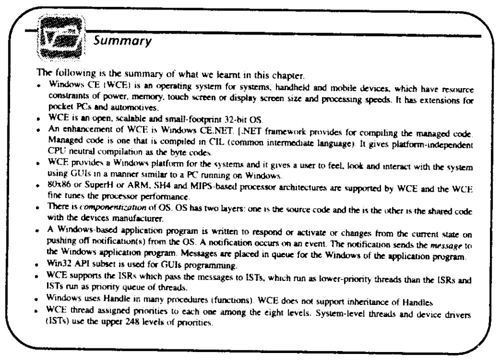


Fig. 1.10 *A SoC embedded system and its common bus with internal ASIPs, internal processors, IPs shared memories and peripheral interfaces*

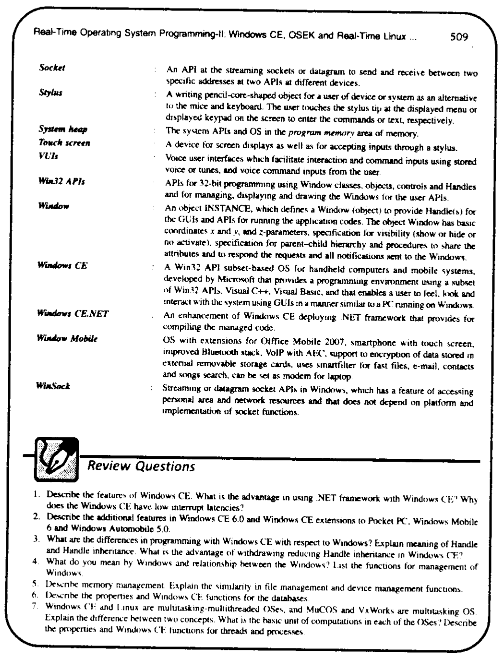
Simple approach with figures to explain complex topic of system on chip for a mobile phone



Summary

The following is the summary of what we learnt in this chapter.

- Windows CE (WCE) is an operating system for systems, handheld and mobile devices, which have resource constraints of power, memory, touch screen or display screen size and processing speeds. It has extensions for pocket PCs and automotives.
- WCE is an open, scalable and small-footprint 32-bit OS.
- An enhancement of WCE is Windows CE.NET. [.NET framework provides for compiling the managed code. Managed code is one that is compiled in CIL (common intermediate language). It gives platform-independent CPU neutral compilation as the byte codes.
- WCE provides a Windows platform for the systems and it gives a user to feel, look and interact with the system using GUIs in a manner similar to a PC running on Windows.
- 80x86 or SuperH or ARM, SH4 and MIPS-based processor architectures are supported by WCE and the WCE fine tunes the processor performance.
- There is *componentization* of OS. OS has two layers: one is the source code and the is the other is the shared code with the devices manufacturer.
- A Windows-based application program is written to respond or activate or changes from the current state on pushing off notification(s) from the OS. A notification occurs on an event. The notification sends the *message* to the Windows application program. Messages are placed in queue for the Windows of the application program.
- Win32 API subset is used for GUIs programming.
- WCE supports the ISRs which pass the messages to ISTs, which run as lower-priority threads than the ISRs and ISTs run as priority queue of threads.
- Windows uses Handle in many procedures (functions). WCE does not support inheritance of Handles
- WCE thread assigned priorities to each one among the eight levels. System-level threads and device drivers (ISTs) use the upper 248 levels of priorities.



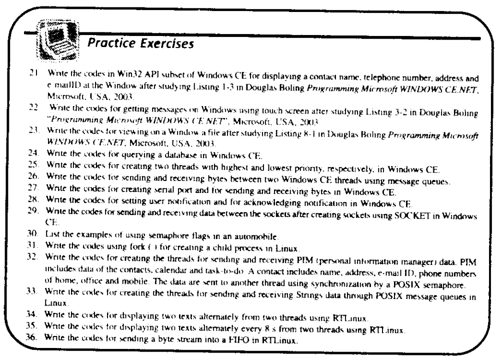
Real-Time Operating System Programming-II: Windows CE, OSEK and Real-Time Linux ... 509

Socket	: An API at the streaming sockets or datagram to send and receive between two specific addresses at two APIs at different devices.
Stylus	: A writing pencil-core-shaped object for a user of device or system as an alternative to the mice and keyboard. The user touches the stylus tip at the displayed menu or displayed keypad on the screen to enter the commands or text, respectively.
System heap	: The system APIs and OS in the *program memory* area of memory.
Touch screen	: A device for screen displays as well as for accepting inputs through a stylus.
VUIs	: Voice user interfaces which facilitate interaction and command inputs using stored voice or tunes, and voice command inputs from the user.
Win32 APIs	: APIs for 32-bit programming using Window classes, objects, controls and Handles and for managing, displaying and drawing the Windows for the user APIs.
Window	: An object INSTANCE, which defines a Window (object) to provide Handle(s) for the GUIs and APIs for running the application codes. The object Window has basic coordinates *x* and *y*, and *z*-parameters, specification for visibility (show or hide or no activate), specification for parent–child hierarchy and procedures to share the attributes and to respond the requests and all notifications sent to the Windows.
Windows CE	: A Win32 API subset-based OS for handheld computers and mobile systems, developed by Microsoft that provides a programming environment using a subset of Win32 APIs, Visual C++, Visual Basic, and that enables a user to feel, look and interact with the system using GUIs in a manner similar to a PC running on Windows.
Windows CE.NET	: An enhancement of Windows CE deploying .NET framework that provides for compiling the managed code.
Window Mobile	OS with extensions for Otffice Mobile 2007, smartphone with touch screen, improved Bluetooth stack, VoIP with AEC, support to encryption of data stored in external removable storage cards, uses smartfilter for fast files, e-mail, contacts and songs search, can be set as modem for laptop.
WinSock	: Streaming or datagram socket APIs in Windows, which has a feature of accessing personal area and network resources and that does not depend on platform and implementation of socket functions.

Review Questions

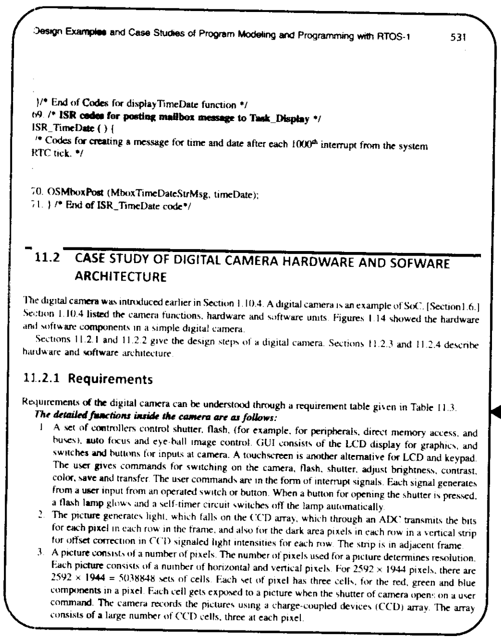
1. Describe the features of Windows CE. What is the advantage in using .NET framework with Windows CE? Why does the Windows CE have low interrupt latencies?
2. Describe the additional features in Windows CE 6.0 and Windows CE extensions to Pocket PC, Windows Mobile 6 and Windows Automobile 5.0.
3. What are the differences in programming with Windows CE with respect to Windows? Explain meaning of Handle and Handle inheritance. What is the advantage of withdrawing reducing Handle inheritance in Windows CE?
4. What do you mean by Windows and relationship between the Windows? List the functions for management of Windows.
5. Describe memory management. Explain the similarity in file management and device management functions.
6. Describe the properties and Windows CE functions for the databases.
7. Windows CE and Linux are multitasking-multithreaded OSes, and MuCOS and VxWorks are multitasking OS. Explain the difference between two concepts. What is the basic unit of computations in each of the OSes? Describe the properties and Windows CE functions for threads and processes.

Summary, keywords and their definitions, review questions and practice exercises in each chapter



Practice Exercises

21. Write the codes in Win32 API subset of Windows CE for displaying a contact name, telephone number, address and e mailID at the Window after studying Listing 1-3 in Douglas Boling *Programming Microsoft WINDOWS CE.NET*, Microsoft, USA, 2003.
22. Write the codes for getting messages on Windows using touch screen after studying Listing 3-2 in Douglas Boling "*Programming Microsoft WINDOWS CE.NET*", Microsoft, USA, 2003.
23. Write the codes for viewing on a Window a file after studying Listing 8-1 in Douglas Boling *Programming Microsoft WINDOWS CE.NET*, Microsoft, USA, 2003.
24. Write the codes for querying a database in Windows CE.
25. Write the codes for creating two threads with highest and lowest priority, respectively, in Windows CE.
26. Write the codes for sending and receiving bytes between two Windows CE threads using message queues.
27. Write the codes for creating serial port and for sending and receiving bytes in Windows CE.
28. Write the codes for setting user notification and for acknowledging notification in Windows CE.
29. Write the codes for sending and receiving data between the sockets after creating sockets using SOCKET in Windows CE.
30. List the examples of using semaphore flags in an automobile.
31. Write the codes using fork ( ) for creating a child process in Linux.
32. Write the codes for creating the threads for sending and receiving PIM (personal information manager) data. PIM includes data of the contacts, calendar and task-to-do. A contact includes name, address, e-mail ID, phone numbers of home, office and mobile. The data are sent to another thread using synchronization by a POSIX semaphore.
33. Write the codes for creating the threads for sending and receiving Strings data through POSIX message queues in Linux.
34. Write the codes for displaying two texts alternately from two threads using RTLinux.
35. Write the codes for displaying two texts alternately every 8 s from two threads using RTLinux.
36. Write the codes for sending a byte stream into a FIFO in RTLinux.



```
}/* End of Codes for displayTimeDate function */
69. /* ISR codes for posting mailbox message to Task_Display */
ISR_TimeDate ( ) {
/* Codes for creating a message for time and date after each 1000th interrupt from the system
RTC tick. */

70. OSMboxPost (MboxTimeDateStrMsg, timeDate);
71. } /* End of ISR_TimeDate code*/
```

11.2 CASE STUDY OF DIGITAL CAMERA HARDWARE AND SOFWARE ARCHITECTURE

The digital camera was introduced earlier in Section 1.10.4. A digital camera is an example of SoC. [Section1.6.] Section 1.10.4 listed the camera functions, hardware and software units. Figures 1.14 showed the hardware and software components in a simple digital camera.

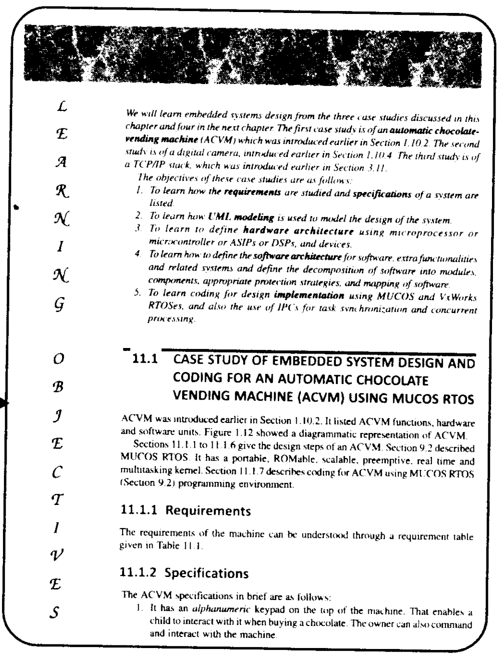
Sections 11.2.1 and 11.2.2 give the design steps of a digital camera. Sections 11.2.3 and 11.2.4 describe hardware and software architecture.

11.2.1 Requirements

Requirements of the digital camera can be understood through a requirement table given in Table 11.3.

The detailed functions inside the camera are as follows:

1. A set of controllers control shutter, flash, (for example, for peripherals, direct memory access, and buses), auto focus and eye-ball image control. GUI consists of the LCD display for graphics, and switches and buttons for inputs at camera. A touchscreen is another alternative for LCD and keypad. The user gives commands for switching on the camera, flash, shutter, adjust brightness, contrast, color, save and transfer. The user commands are in the form of interrupt signals. Each signal generates from a user input from an operated switch or button. When a button for opening the shutter is pressed, a flash lamp glows and a self-timer circuit switches off the lamp automatically.
2. The picture generates light, which falls on the CCD array, which through an ADC transmits the bits for each pixel in each row in the frame, and also for the dark area pixels in each row in a vertical strip for offset correction in CCD signaled light intensities for each row. The strip is in adjacent frame.
3. A picture consists of a number of pixels. The number of pixels used for a picture determines resolution. Each picture consists of a number of horizontal and vertical pixels. For 2592 × 1944 pixels, there are 2592 × 1944 = 5038848 sets of cells. Each set of pixel has three cells, for the red, green and blue components in a pixel. Each cell gets exposed to a picture when the shutter of camera opens on a user command. The camera records the pictures using a charge-coupled devices (CCD) array. The array consists of a large number of CCD cells, three at each pixel.



L E A R N I N G O B J E C T I V E S

*We will learn embedded systems design from the three case studies discussed in this chapter and four in the next chapter. The first case study is of an **automatic chocolate-vending machine** (ACVM) which was introduced earlier in Section 1.10.2. The second study is of a digital camera, introduced earlier in Section 1.10.4. The third study is of a TCP/IP stack, which was introduced earlier in Section 3.11.*

The objectives of these case studies are as follows:

1. *To learn how **the requirements** are studied and **specifications** of a system are listed.*
2. *To learn how **UML modeling** is used to model the design of the system.*
3. *To learn to define **hardware architecture** using microprocessor or microcontroller or ASIPs or DSPs, and devices.*
4. *To learn how to define the **software architecture** for software, extra functionalities and related systems and define the decomposition of software into modules, components, appropriate protection strategies, and mapping of software.*
5. *To learn coding for design **implementation** using MUCOS and VxWorks RTOSes, and also the use of IPCs for task synchronization and concurrent processing.*

11.1 CASE STUDY OF EMBEDDED SYSTEM DESIGN AND CODING FOR AN AUTOMATIC CHOCOLATE VENDING MACHINE (ACVM) USING MUCOS RTOS

ACVM was introduced earlier in Section 1.10.2. It listed ACVM functions, hardware and software units. Figure 1.12 showed a diagrammatic representation of ACVM.

Sections 11.1.1 to 11.1.6 give the design steps of an ACVM. Section 9.2 described MUCOS RTOS. It has a portable, ROMable, scalable, preemptive, real time and multitasking kernel. Section 11.1.7 describes coding for ACVM using MUCOS RTOS (Section 9.2) programming environment.

11.1.1 Requirements

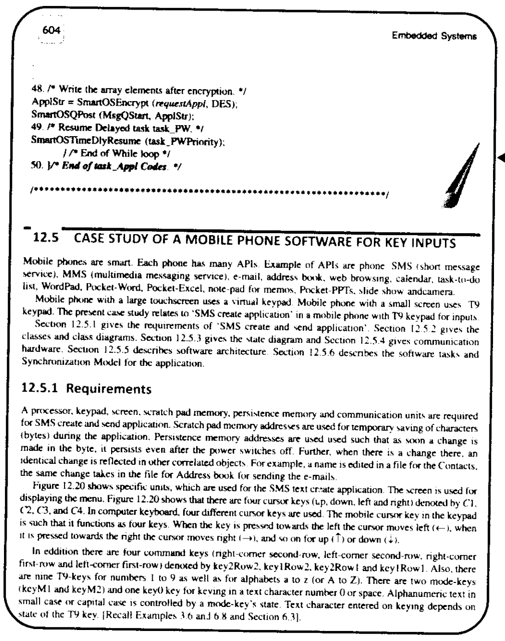
The requirements of the machine can be understood through a requirement table given in Table 11.1.

11.1.2 Specifications

The ACVM specifications in brief are as follows:

1. It has an *alphanumeric* keypad on the top of the machine. That enables a child to interact with it when buying a chocolate. The owner can also command and interact with the machine.

Explains modeling of programs and software engineering practices for system design by case studies of systems for automatic chocolate vending machine, digital camera, TCP/IP stack creation, robot orchestra, automatic cruise control, smart card and mobile phone



```
48. /* Write the array elements after encryption. */
ApplStr = SmartOSEncrypt (requestAppl, DES);
SmartOSQPost (MsgQStart, ApplStr);
49. /* Resume Delayed task task_PW. */
SmartOSTimeDlyResume (task_PWPriority);
        } /* End of While loop */
50. }/* End of task_Appl Codes. */

/**************************************************************/
```

12.5 CASE STUDY OF A MOBILE PHONE SOFTWARE FOR KEY INPUTS

Mobile phones are smart. Each phone has many APIs. Example of APIs are phone SMS (short message service), MMS (multimedia messaging service), e-mail, address book, web browsing, calendar, task-to-do list, WordPad, Pocket-Word, Pocket-Excel, note-pad for memos, Pocket-PPTs, slide show andcamera.

Mobile phone with a large touchscreen uses a virtual keypad. Mobile phone with a small screen uses T9 keypad. The present case study relates to 'SMS create application' in a mobile phone with T9 keypad for inputs.

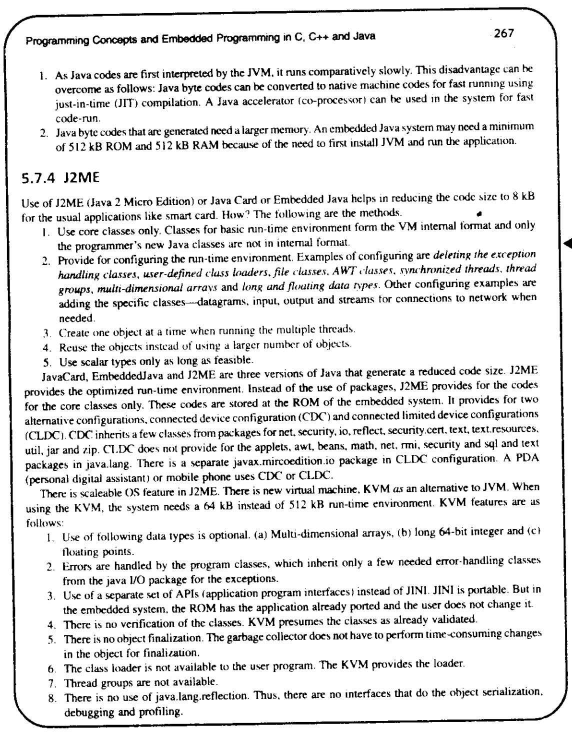
Section 12.5.1 gives the requirements of 'SMS create and send application'. Section 12.5.2 gives the classes and class diagrams. Section 12.5.3 gives the state diagram and Section 12.5.4 gives communication hardware. Section 12.5.5 describes software architecture. Section 12.5.6 describes the software tasks and Synchronization Model for the application.

12.5.1 Requirements

A processor, keypad, screen, scratch pad memory, persistence memory and communication units are required for SMS create and send application. Scratch pad memory addresses are used for temporary saving of characters (bytes) during the application. Persistence memory addresses are used used such that as soon a change is made in the byte, it persists even after the power switches off. Further, when there is a change there, an identical change is reflected in other correlated objects. For example, a name is edited in a file for the Contacts, the same change takes in the file for Address book for sending the e-mails.

Figure 12.20 shows specific units, which are used for the SMS text create application. The screen is used for displaying the menu. Figure 12.20 shows that there are four cursor keys (up, down, left and right) denoted by C1, C2, C3, and C4. In computer keyboard, four different cursor keys are used. The mobile cursor key in the keypad is such that it functions as four keys. When the key is pressed towards the left the cursor moves left (←), when it is pressed towards the right the cursor moves right (→), and so on for up (↑) or down (↓).

In eddition there are four command keys (right-corner second-row, left-corner second-row, right-corner first-row and left-corner first-row) denoted by key2Row2, key1Row2, key2Row1 and key1Row1. Also, there are nine T9-keys for numbers 1 to 9 as well as for alphabets a to z (or A to Z). There are two mode-keys (keyM1 and keyM2) and one key0 key for keying in a text character number 0 or space. Alphanumeric text in small case or capital case is controlled by a mode-key's state. Text character entered on keying depends on state of the T9 key. [Recall Examples 3.6 and 6.8 and Section 6.3].



1. As Java codes are first interpreted by the JVM, it runs comparatively slowly. This disadvantage can be overcome as follows: Java byte codes can be converted to native machine codes for fast running using just-in-time (JIT) compilation. A Java accelerator (co-processor) can be used in the system for fast code-run.
2. Java byte codes that are generated need a larger memory. An embedded Java system may need a minimum of 512 kB ROM and 512 kB RAM because of the need to first install JVM and run the application.

5.7.4 J2ME

Use of J2ME (Java 2 Micro Edition) or Java Card or Embedded Java helps in reducing the code size to 8 kB for the usual applications like smart card. How? The following are the methods.

1. Use core classes only. Classes for basic run-time environment form the VM internal format and only the programmer's new Java classes are not in internal format.
2. Provide for configuring the run-time environment. Examples of configuring are *deleting the exception handling classes*, *user-defined class loaders*, *file classes*, *AWT classes*, *synchronized threads*, *thread groups*, *multi-dimensional arrays* and *long and floating data types*. Other configuring examples are adding the specific classes—datagrams, input, output and streams for connections to network when needed.
3. Create one object at a time when running the multiple threads.
4. Reuse the objects instead of using a larger number of objects.
5. Use scalar types only as long as feasible.

JavaCard, EmbeddedJava and J2ME are three versions of Java that generate a reduced code size. J2ME provides the optimized run-time environment. Instead of the use of packages, J2ME provides for the codes for the core classes only. These codes are stored at the ROM of the embedded system. It provides for two alternative configurations, connected device configuration (CDC) and connected limited device configurations (CLDC). CDC inherits a few classes from packages for net, security, io, reflect, security.cert, text, text.resources, util, jar and zip. CLDC does not provide for the applets, awt, beans, math, net, rmi, security and sql and text packages in java.lang. There is a separate javax.mircoedition.io package in CLDC configuration. A PDA (personal digital assistant) or mobile phone uses CDC or CLDC.

There is scaleable OS feature in J2ME. There is new virtual machine, KVM *as* an alternative to JVM. When using the KVM, the system needs a 64 kB instead of 512 kB run-time environment. KVM features are as follows:

1. Use of following data types is optional. (a) Multi-dimensional arrays, (b) long 64-bit integer and (c) floating points.
2. Errors are handled by the program classes, which inherit only a few needed error-handling classes from the java I/O package for the exceptions.
3. Use of a separate set of APIs (application program interfaces) instead of JINI. JINI is portable. But in the embedded system, the ROM has the application already ported and the user does not change it.
4. There is no verification of the classes. KVM presumes the classes as already validated.
5. There is no object finalization. The garbage collector does not have to perform time-consuming changes in the object for finalization.
6. The class loader is not available to the user program. The KVM provides the loader.
7. Thread groups are not available.
8. There is no use of java.lang.reflection. Thus, there are no interfaces that do the object serialization, debugging and profiling.

Simple way of point-wise presentation of the details by using lists and tables

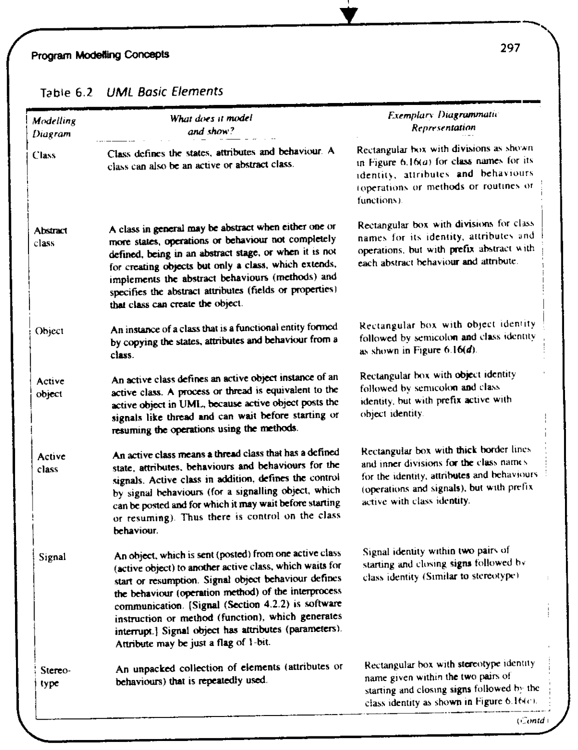
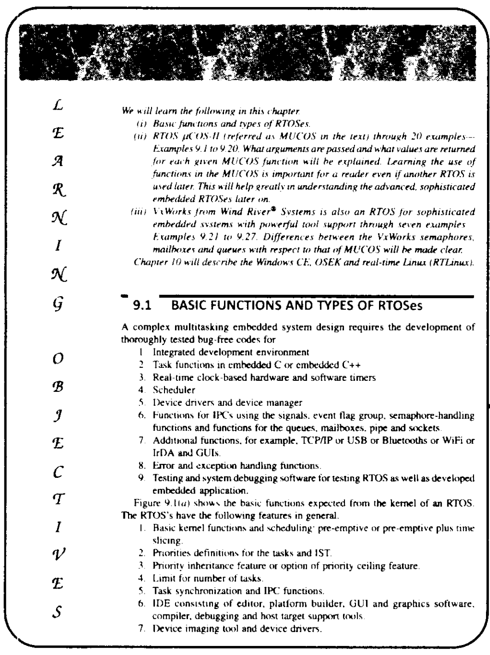


Table 6.2 *UML Basic Elements*

Modelling Diagram	*What does it model and show?*	*Exemplary Diagrammatic Representation*
Class	Class defines the states, attributes and behaviour. A class can also be an active or abstract class.	Rectangular box with divisions as shown in Figure 6.16(*a*) for class names for its identity, attributes and behaviours (operations or methods or routines or functions).
Abstract class	A class in general may be abstract when either one or more states, operations or behaviour not completely defined, being in an abstract stage, or when it is not for creating objects but only a class, which extends, implements the abstract behaviours (methods) and specifies the abstract attributes (fields or properties) that class can create the object.	Rectangular box with divisions for class names for its identity, attributes and operations, but with prefix abstract with each abstract behaviour and attribute.
Object	An instance of a class that is a functional entity formed by copying the states, attributes and behaviour from a class.	Rectangular box with object identity followed by semicolon and class identity as shown in Figure 6.16(***d***).
Active object	An active class defines an active object instance of an active class. A process or thread is equivalent to the active object in UML, because active object posts the signals like thread and can wait before starting or resuming the operations using the methods.	Rectangular box with object identity followed by semicolon and class identity, but with prefix active with object identity.
Active class	An active class means a thread class that has a defined state, attributes, behaviours and behaviours for the signals. Active class in addition, defines the control by signal behaviours (for a signalling object, which can be posted and for which it may wait before starting or resuming). Thus there is control on the class behaviour.	Rectangular box with thick border lines and inner divisions for the class names for the identity, attributes and behaviours (operations and signals), but with prefix active with class identity.
Signal	An object, which is sent (posted) from one active class (active object) to another active class, which waits for start or resumption. Signal object behaviour defines the behaviour (operation method) of the interprocess communication. [Signal (Section 4.2.2) is software instruction or method (function), which generates interrupt.] Signal object has attributes (parameters). Attribute may be just a flag of 1-bit.	Signal identity within two pairs of starting and closing signs followed by class identity (Similar to stereotype)
Stereo-type	An unpacked collection of elements (attributes or behaviours) that is repeatedly used.	Rectangular box with stereotype identity name given within the two pairs of starting and closing signs followed by the class identity as shown in Figure 6.16(*c*).

(*Contd*)



L E A R N I N G O B J E C T I V E S

We will learn the following in this chapter.

(i) Basic functions and types of RTOSes.

(ii) RTOS μCOS-II (referred as MUCOS in the text) through 20 examples—Examples 9.1 to 9.20. What arguments are passed and what values are returned for each given MUCOS function will be explained. Learning the use of functions in the MUCOS is important for a reader even if another RTOS is used later. This will help greatly in understanding the advanced, sophisticated embedded RTOSes later on.

(iii) VxWorks from Wind River® Systems is also an RTOS for sophisticated embedded systems with powerful tool support through seven examples Examples 9.21 to 9.27. Differences between the VxWorks semaphores, mailboxes and queues with respect to that of MUCOS will be made clear.

Chapter 10 will describe the Windows CE, OSEK and real-time Linux (RTLinux).

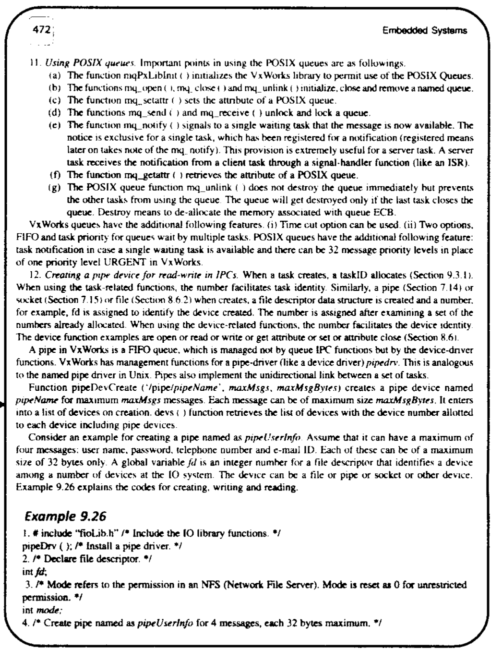
9.1 BASIC FUNCTIONS AND TYPES OF RTOSes

A complex multitasking embedded system design requires the development of thoroughly tested bug-free codes for

1. Integrated development environment
2. Task functions in embedded C or embedded C++
3. Real-time clock-based hardware and software timers
4. Scheduler
5. Device drivers and device manager
6. Functions for IPCs using the signals, event flag group, semaphore-handling functions and functions for the queues, mailboxes, pipe and sockets.
7. Additional functions, for example, TCP/IP or USB or Bluetooths or WiFi or IrDA and GUIs.
8. Error and exception handling functions.
9. Testing and system debugging software for testing RTOS as well as developed embedded application.

Figure 9.1(*a*) shows the basic functions expected from the kernel of an RTOS. The RTOS's have the following features in general.

1. Basic kernel functions and scheduling: pre-emptive or pre-emptive plus time slicing.
2. Priorities definitions for the tasks and IST.
3. Priority inheritance feature or option of priority ceiling feature.
4. Limit for number of tasks.
5. Task synchronization and IPC functions.
6. IDE consisting of editor, platform builder, GUI and graphics software, compiler, debugging and host target support tools.
7. Device imaging tool and device drivers.



472 Embedded Systems

11. *Using POSIX queues.* Important points in using the POSIX queues are as followings.
 (a) The function mqPxLibInit ( ) initializes the VxWorks library to permit use of the POSIX Queues.
 (b) The functions mq_open ( ), mq_close ( ) and mq_unlink ( ) initialize, close and remove a named queue.
 (c) The function mq_setattr ( ) sets the attribute of a POSIX queue.
 (d) The functions mq_send ( ) and mq_receive ( ) unlock and lock a queue.
 (e) The function mq_notify ( ) signals to a single waiting task that the message is now available. The notice is exclusive for a single task, which has been registered for a notification (registered means later on takes note of the mq_notify). This provision is extremely useful for a server task. A server task receives the notification from a client task through a signal-handler function (like an ISR).
 (f) The function mq_getattr ( ) retrieves the attribute of a POSIX queue.
 (g) The POSIX queue function mq_unlink ( ) does not destroy the queue immediately but prevents the other tasks from using the queue. The queue will get destroyed only if the last task closes the queue. Destroy means to de-allocate the memory associated with queue ECB.

VxWorks queues have the additional following features. (i) Time cut option can be used. (ii) Two options, FIFO and task priority for queues wait by multiple tasks. POSIX queues have the additional following feature: task notification in case a single waiting task is available and there can be 32 message priority levels in place of one priority level URGENT in VxWorks.

12. *Creating a pipe device for read-write in IPCs.* When a task creates, a taskID allocates (Section 9.3.1). When using the task-related functions, the number facilitates task identity. Similarly, a pipe (Section 7.14) or socket (Section 7.15) or file (Section 8.6.2) when creates, a file descriptor data structure is created and a number, for example, fd is assigned to identify the device created. The number is assigned after examining a set of the numbers already allocated. When using the device-related functions, the number facilitates the device identity. The device function examples are open or read or write or get attribute or set or attribute close (Section 8.6).

A pipe in VxWorks is a FIFO queue, which is managed not by queue IPC functions but by the device-driver functions. VxWorks has management functions for a pipe-driver (like a device driver) *pipedrv*. This is analogous to the named pipe driver in Unix. Pipes also implement the unidirectional link between a set of tasks.

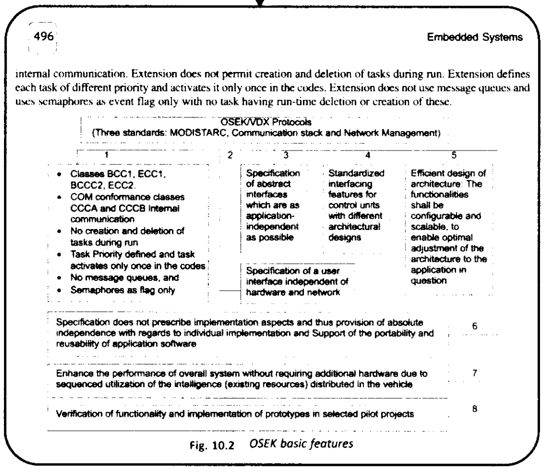
Function pipeDevCreate ('/pipe/*pipeName*', *maxMsgs*, *maxMsgBytes*) creates a pipe device named *pipeName* for maximum *maxMsgs* messages. Each message can be of maximum size *maxMsgBytes*. It enters into a list of devices on creation. devs ( ) function retrieves the list of devices with the device number allotted to each device including pipe devices.

Consider an example for creating a pipe named as *pipeUserInfo*. Assume that it can have a maximum of four messages: user name, password, telephone number and e-mail ID. Each of these can be of a maximum size of 32 bytes only. A global variable *fd* is an integer number for a file descriptor that identifies a device among a number of devices at the IO system. The device can be a file or pipe or socket or other device. Example 9.26 explains the codes for creating, writing and reading.

Example 9.26

```
1. # include "fioLib.h" /* Include the IO library functions. */
pipeDrv ( ); /* Install a pipe driver. */
2. /* Declare file descriptor. */
int fd;
 3. /* Mode refers to the permission in an NFS (Network File Server). Mode is reset as 0 for unrestricted
permission. */
int mode;
4. /* Create pipe named as pipeUserInfo for 4 messages, each 32 bytes maximum. */
```

Comprehensive explanation with coding examples for learning the widely used RTOSes- mCOS-II, VxWorks, Windows CE, OSEK and Real Time Linux



496 Embedded Systems

internal communication. Extension does not permit creation and deletion of tasks during run. Extension defines each task of different priority and activates it only once in the codes. Extension does not use message queues and uses semaphores as event flag only with no task having run-time deletion or creation of these.

OSEK/VDX Protocols
(Three standards: MODISTARC, Communication stack and Network Management)

1
- Classes BCC1, ECC1, BCCC2, ECC2.
- COM conformance classes CCCA and CCCB internal communication
- No creation and deletion of tasks during run
- Task Priority defined and task activates only once in the codes
- No message queues, and
- Semaphores as flag only

2
Specification of a user interface independent of hardware and network

3
Specification of abstract interfaces which are as application-independent as possible

4
Standardized interfacing features for control units with different architectural designs

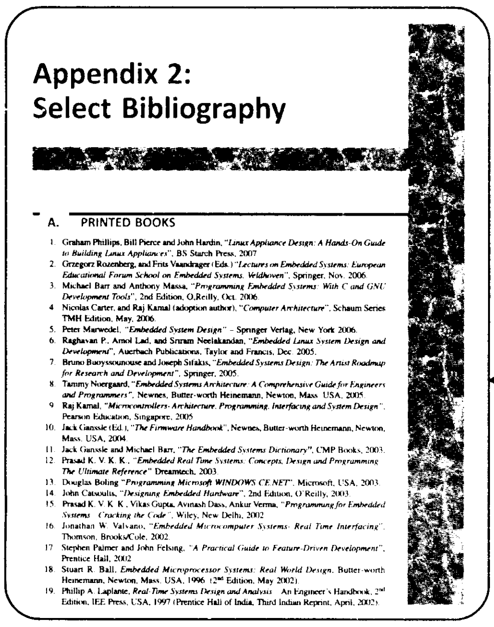
5
Efficient design of architecture: The functionalities shall be configurable and scalable, to enable optimal adjustment of the architecture to the application in question

6
Specification does not prescribe implementation aspects and thus provision of absolute independence with regards to individual implementation and Support of the portability and reusability of application software

7
Enhance the performance of overall system without requiring additional hardware due to sequenced utilization of the intelligence (existing resources) distributed in the vehicle

8
Verification of functionality and implementation of prototypes in selected pilot projects

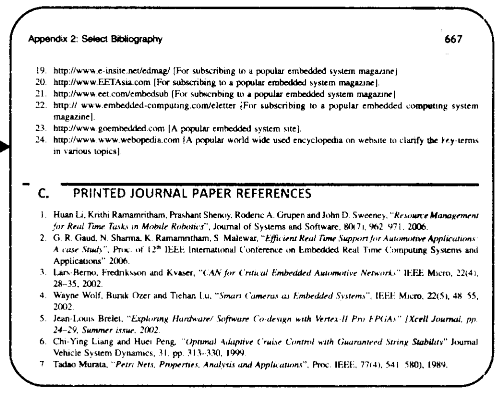
Fig. 10.2 *OSEK basic features*



Appendix 2:
Select Bibliography

A. PRINTED BOOKS

1. Graham Phillips, Bill Pierce and John Hardin, "*Linux Appliance Design: A Hands-On Guide to Building Linux Appliances*", BS Starch Press, 2007
2. Grzegorz Rozenberg, and Frits Vaandrager (Eds.) "*Lectures on Embedded Systems: European Educational Forum School on Embedded Systems, Veldhoven*", Springer, Nov. 2006.
3. Michael Barr and Anthony Massa, "*Programming Embedded Systems: With C and GNU Development Tools*", 2nd Edition, O,Reilly, Oct. 2006.
4. Nicolas Carter, and Raj Kamal (adoption author), "*Computer Architecture*", Schaum Series TMH Edition, May, 2006.
5. Peter Marwedel, "*Embedded System Design*" – Springer Verlag, New York 2006.
6. Raghavan P., Amol Lad, and Sriram Neelakandan, "*Embedded Linux System Design and Development*", Auerbach Publications, Taylor and Francis, Dec. 2005.
7. Bruno Buoyssounouse and Joseph Sifakis, "*Embedded Systems Design: The Artist Roadmap for Research and Development*", Springer, 2005.
8. Tammy Noergaard, "*Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*", Newnes, Butter-worth Heinemann, Newton, Mass. USA, 2005.
9. Raj Kamal, "*Microcontrollers- Architecture, Programming, Interfacing and System Design*", Pearson Education, Singapore, 2005
10. Jack Ganssle (Ed.), "*The Firmware Handbook*", Newnes, Butter-worth Heinemann, Newton, Mass. USA, 2004.
11. Jack Ganssle and Michael Barr, "*The Embedded Systems Dictionary*", CMP Books, 2003.
12. Prasad K. V. K. K., "*Embedded Real Time Systems: Concepts, Design and Programming The Ultimate Reference*" Dreamtech, 2003.
13. Douglas Boling "*Programming Microsoft WINDOWS CE.NET*", Microsoft, USA, 2003.
14. John Catsoulis, "*Designing Embedded Hardware*", 2nd Edition, O'Reilly, 2003.
15. Prasad K. V. K. K., Vikas Gupta, Avinash Dass, Ankur Verma, "*Programming for Embedded Systems Cracking the Code*", Wiley, New Delhi, 2002
16. Jonathan W. Valvano, "*Embedded Microcomputer Systems- Real Time Interfacing*", Thomson, Brooks/Cole, 2002.
17. Stephen Palmer and John Felsing, "*A Practical Guide to Feature-Driven Development*", Prentice Hall, 2002
18. Stuart R. Ball, *Embedded Microprocessor Systems: Real World Design*, Butter-worth Heinemann, Newton, Mass. USA, 1996 (2nd Edition, May 2002).
19. Phillip A. Laplante, *Real-Time Systems Design and Analysis* An Engineer's Handbook, 2nd Edition, IEE Press, USA, 1997 (Prentice Hall of India, Third Indian Reprint, April, 2002).



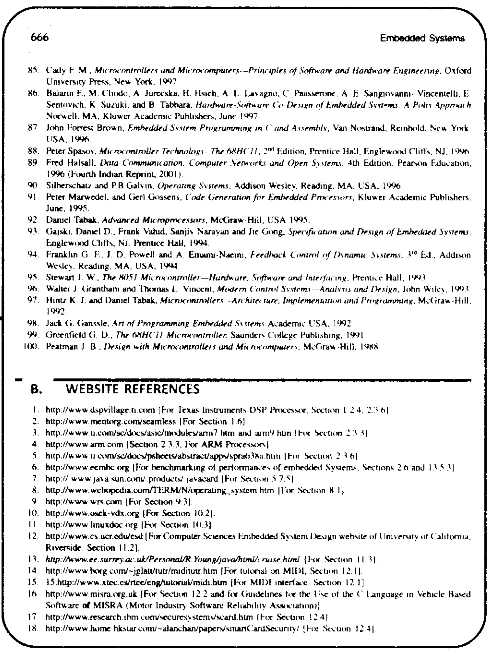
Appendix 2: Select Bibliography 667

19. http://www.e-insite.net/edmag/ [For subscribing to a popular embedded system magazine]
20. http://www.EETAsia.com [For subscribing to a popular embedded system magazine].
21. http://www.eet.com/embedsub [For subscribing to a popular embedded system magazine]
22. http:// www.embedded-computing.com/eletter [For subscribing to a popular embedded computing system magazine].
23. http://www.goembedded.com [A popular embedded system site].
24. http://www.www.webopedia.com [A popular world wide used encyclopedia on website to clarify the key-terms in various topics].

C. PRINTED JOURNAL PAPER REFERENCES

1. Huan Li, Krithi Ramamritham, Prashant Shenoy, Roderic A. Grupen and John D. Sweeney, "*Resource Management for Real Time Tasks in Mobile Robotics*", Journal of Systems and Software, 80(7), 962–971, 2006.
2. G. R. Gaud, N. Sharma, K. Ramamritham, S. Malewar, "*Efficient Real Time Support for Automotive Applications: A case Study*", Proc. of 12th IEEE International Conference on Embedded Real Time Computing Systems and Applications" 2006.
3. Lars-Berno, Fredriksson and Kvaser, "*CAN for Critical Embedded Automotive Networks*" IEEE Micro, 22(4), 28–35, 2002.
4. Wayne Wolf, Burak Ozer and Tiehan Lu, "*Smart Cameras as Embedded Systems*", IEEE Micro, 22(5), 48–55, 2002.
5. Jean-Louis Brelet, "*Exploring Hardware/ Software Co-design with Vertex-II Pro FPGAs" [Xcell Journal, pp. 24–29, Summer issue, 2002.*
6. Chi-Ying Liang and Huei Peng, "*Optimal Adaptive Cruise Control with Guaranteed String Stability*" Journal Vehicle System Dynamics, 31, pp. 313–330, 1999.
7. Tadao Murata, "*Petri Nets, Properties, Analysis and Applications*", Proc. IEEE, 77(4), 541–580), 1989.

Detailed selected bibliography of books, journal references and important web links at end of the book to facilitate building a startup library for references and further studies in Embedded Systems



666 Embedded Systems

85. Cady F. M., *Microcontrollers and Microcomputers—Principles of Software and Hardware Engineering*, Oxford University Press, New York, 1997
86. Balarin F., M. Chiodo, A. Jurecska, H. Hsieh, A. L. Lavagno, C. Paasserone, A. E. Sangiovanni- Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems: A Polis Approach* Norwell, MA, Kluwer Academic Publishers, June 1997.
87. John Forrest Brown, *Embedded System Programming in C and Assembly*, Van Nostrand, Reinhold, New York, USA, 1996.
88. Peter Spasov, *Microcontroller Technology- The 68HC11*, 2nd Edition, Prentice Hall, Englewood Cliffs, NJ, 1996.
89. Fred Halsall, *Data Communication, Computer Networks and Open Systems*, 4th Edition, Pearson Education, 1996 (Fourth Indian Reprint, 2001).
90. Silberschatz and P.B.Galvin, *Operating Systems*, Addison Wesley, Reading, MA, USA, 1996.
91. Peter Marwedel, and Gert Gossens, *Code Generation for Embedded Processors*, Kluwer Academic Publishers, June, 1995.
92. Daniel Tabak, *Advanced Microprocessors*, McGraw-Hill, USA 1995.
93. Gajski, Daniel D., Frank Vahid, Sanjiv Narayan and Jie Gong, *Specification and Design of Embedded Systems*, Englewood Cliffs, NJ, Prentice Hall, 1994.
94. Franklin G. F., J. D. Powell and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 3rd Ed., Addison Wesley, Reading, MA, USA, 1994.
95. Stewart J. W., *The 8051 Microcontroller—Hardware, Software and Interfacing*, Prentice Hall, 1993
96. Walter J. Grantham and Thomas L. Vincent, *Modern Control Systems—Analysis and Design*, John Wiley, 1993
97. Hintz K. J. and Daniel Tabak, *Microcontrollers –Architecture, Implementation and Programming*, McGraw-Hill, 1992.
98. Jack G. Ganssle, *Art of Programming Embedded Systems* Academic USA, 1992
99. Greenfield G. D., *The 68HC11 Microcontroller*, Saunders College Publishing, 1991
100. Peatman J. B., *Design with Microcontrollers and Microcomputers*, McGraw-Hill, 1988

B. WEBSITE REFERENCES

1. http://www.dspvillage.ti.com [For Texas Instruments DSP Processor, Section 1.2.4, 2.3.6].
2. http://www.mentorg.com/seamless [For Section 1.6]
3. http://www.ti.com/sc/docs/asic/modules/arm7.htm and arm9.htm [For Section 2.3.3]
4. http://www.arm.com [Section 2.3.3, For ARM Processors].
5. http://www.ti.com/sc/docs/psheets/abstract/apps/spra638a.htm [For Section 2.3.6]
6. http://www.eembc.org [For benchmarking of performances of embedded Systems, Sections 2.6 and 13.5.3]
7. http:// www.java.sun.com/ products/ javacard [For Section 5.7.5]
8. http://www.webopedia.com/TERM/N/operating_system.htm [For Section 8.1]
9. http://www.wrs.com [For Section 9.3].
10. http://www.osek-vdx.org [For Section 10.2].
11. http://www.linuxdoc.org [For Section 10.3]
12. http://www.cs.ucr.edu/esd [For Computer Sciences Embedded System Design website of University of California, Riverside, Section 11.2].
13. *http://www.ee.surrey.ac.uk/Personal/R.Young/java/html/cruise.html* [For Section 11.3].
14. http://www.borg.com/~jglatt/tutr/miditutr.htm [For tutorial on MIDI, Section 12.1].
15. 15.http://www.xtec.es/rtee/eng/tutorial/midi.htm [For MIDI interface, Section 12.1].
16. http://www.misra.org.uk [For Section 12.2 and for Guidelines for the Use of the C Language in Vehicle Based Software of MISRA (Motor Industry Software Reliability Association)]
17. http://www.research.ibm.com/securesystems/scard.htm [For Section 12.4]
18. http://www.home.hkstar.com/~alanchan/papers/smartCardSecurity/ [For Section 12.4].

Introduction to Embedded Systems

1

LEARNING OBJECTIVES

Section 1.1
*Definitions of **system** and **embedded system***

Section 1.2
The processing unit of an embedded system consists of
1. *A processor*
2. *Commonly used microprocessors*
3. *Application-specific instruction set processors (ASIPs), microcontrollers, DSPs and others*
4. *Single purpose processors*

Section 1.3
The hardware unit of an embedded system consists of
1. *An embedded system power source with controlled power-dissipation*
2. *A clock oscillator circuit and clocking unit that lets a processor execute instructions*
3. *Timers and a real time clock (RTC) for various timing needs of the system*
4. *Reset circuit and watchdog timer*
5. *System and external memories*
6. *System input output (IO) ports, serial, parallel and wireless communication, serial Universal Asynchronous Receiver and Transmitter (UART) and other port protocols and buses*
7. *Devices such as Digital to Analog Converter (DAC) using Pulse Width Modulation (PWM), Analog to Digital Converter (ADC), Light Emitting Diode (LED) and Liquid Crystal Display (LCD) units, keypad and keyboard, touch screen, pulse dialer, modem and transceiver*
8. *Multiplexers, demultiplexers, decoder for interfacing of the devices and buses*

L E A R N I N G O B J E C T I V E S

9. *Interrupt controller (handler)*

Section 1.4

1. *Languages that are used to develop* **embedded software** *for a system*
2. *Program models*
3. *Multitasking using an operating system (OS), system device drivers, device management and real time operating system (RTOS)*
4. *Software tools for system design*

Section 1.5

Examples of applications of embedded systems

Section 1.6

Designing an embedded system on a VLSI chip

1. *Embedded SoC (System on Chip) and examples of its applications*
2. *Uses of Application Specific Instruction Set Processor (ASIP) and Intellectual Property (IP) core*
3. *Field Programmable Gate Array (FPGA) core with single or multiple processor units on an ASIC chip*

Section 1.7

The complex system consists of

1. *Embedded microprocessors or GPPs in complex systems*
2. *Embedding ASIPs, microcontrollers, DSPs, media and network processors*
3. *Embedding application-specific system processors (ASSPs)*
4. *Embedding multiple processors in systems*

Section 1.8

The design process has

1. *Challenges in embedded system design*
2. *Design metrics optimization*
3. *Co-design of hardware and software components*

Section 1.9

The system design formalism is defined

Section 1.10

The design of embedded hardware and software in an automatic chocolate vending machine, smart card, digital-camera, mobile phone, mobile computer and robot are given as examples

Section 1.11

Classification of embedded systems into three types

Section 1.12

Skills needed to design an embedded system

1.1 EMBEDDED SYSTEMS

1.1.1 System

A system is a way of working, organizing or doing one or many tasks according to a fixed plan, program, or set of rules. A system is also an arrangement in which all its units assemble and work together according to the plan or program.

Consider a watch. It is a time-display system. Its parts are its hardware, needles and battery with the beautiful dial, chassis and strap. *These parts organize to show* the real time every second and continuously update the time every second. The system-program updates the display using three needles after each second. *It follows a set of rules*. Some of these rules are as follows: (i) All needles move only clockwise. (ii) A thin and long needle rotates every second such that it returns to same position after a minute. (iii) A long needle rotates every minute such that it returns to same position after an hour. (iv) A short needle rotates every hour such that it returns to same position after twelve hours. (v) All three needles return to the same inclination after twelve hours each day.

Consider a washing machine. It is an automatic clothes-washing system. The important hardware parts include its status display panel, the switches and dials for user-defined programming, a motor to rotate or spin, its power supply and control unit, an inner water-level sensor, a solenoid valve for letting water in and another valve for letting water drain out. *These parts organize* to wash clothes automatically according to a program preset by a user. *The system-program* is activated to wash the dirty clothes placed in a tank, which rotates or spins in preprogrammed steps and stages. *It follows a set of rules*. Some of these rules are as follows: (i) Follow the steps strictly in the following sequence. Step I: Wash by spinning the motor according to a programmed period. Step II: Rinse in fresh water after draining out the dirty water, and rinse a second time if the system is not programmed in water-saving mode. Step III: After draining out the water completely, spin the motor fast for a programmed period for drying by centrifuging out water from the clothes. Step IV: Show the wash-over status by a blinking display. Sound the alarm for a minute to signal that the wash cycle is complete. (ii) At each step, display the process stage of the system. (iii) In case of an interruption, execute only the remaining part of the program, starting from the position when the process was interrupted. There can be no repetition from Step I unless the user resets the system by inserting another set of clothes and resets the program.

1.1.2 Embedded System

Definition One of the definitions of *embedded system* is as follows:

"*An embedded system is a system that has embedded software and computer-hardware, which makes it a system dedicated for an application(s) or specific part of an application or product or a part of a larger system.*"

Embedded systems have been defined in books published recently in several ways. Given below is a series of definitions from others in the field:

Wayne Wolf author of *Computers as Components – Principles of Embedded Computing System Design:* "What is an *embedded computing system*? Loosely defined, it is any device that includes a programmable computer but is not itself intended to be a general-purpose computer" and "a fax machine or a clock built from a microprocessor is an embedded computing system".

Todd D. Morton author of *Embedded Microcontrollers*: "*Embedded Systems* are electronic systems that contain a microprocessor or microcontroller, but we do not think of them as computers—the computer is hidden or embedded in the system."

David E. Simon author of *An Embedded Software Primer*: "People use the term *embedded system* to mean any computer system hidden in any of these products."

Tim Wilmshurst author of *An Introduction to the Design of Small Scale Embedded Systems* with examples from PIC, 80C51 and 68HC05/08 microcontrollers: (1) "An embedded system is a system whose principal function is not computational, but which is controlled by a computer embedded within it. The computer is likely to be a microprocessor or microcontroller. The word embedded implies that it lies inside the overall system, hidden from view, forming an integral part of [the] greater whole". (2) "An embedded system is a microcontroller-based, software-driven, reliable, real time control system, autonomous, or human- or network-interactive, operating on diverse physical variables and in diverse environments, and sold into a competitive and cost-conscious market".

A computer is a system that has the following or more components.

1. A microprocessor
2. A large memory of the following two kinds:
   (a) Primary memory (*semiconductor* memories: Random Access Memory (RAM), Read Only Memory (ROM) and fast accessible caches)
   (b) Secondary memory [(*magnetic* memory located in hard disks, diskettes and cartridge tapes, *optical* memory in CD-ROMs or memory sticks (in mobile computers)] using which different user programs can be loaded into the primary memory and run
3. I/O units such as touch screen, modem, fax cum modem, etc.
4. Input units such as keyboard, mice, digitizer, scanner, etc.
5. Output units such as an LCD screen, video monitor, printer, etc.
6. Networking units such as an Ethernet card, front-end processor-based server, bus drivers, etc.
7. An operating system (OS) that has general purpose user and application software in the secondary memory

An embedded system is a system that has three main components embedded into it:

1. It embeds hardware similar to a computer. Figure 1.1 shows the units in the hardware of an embedded system. As its software usually embeds in the ROM or flash memory, it usually do not need a secondary hard disk and CD memory as in a computer
2. It embeds main application software. The application software may concurrently perform a series of tasks or processes or threads
3. It embeds a real-time operating system (RTOS) that supervises the application software running on hardware and organizes access to a resource according to the priorities of tasks in the system. It provides a mechanism to let the processor run a process as scheduled and context-switch between the various processes. (The concept of process, thread and task explained later in Sections 7.1 to 7.3.) It sets the rules during the execution of the application software. (A small-scale embedded system may not embed the RTOS.)

Characteristics An embedded system is characterized by the following: (1) Real-time and multirate operations define the ways in which the system works, reacts to events, interrupts and schedules the system's functioning in real time. It does so by following a plan to control latencies and to meet deadlines. (Latency refers to the waiting period between running the codes of a task or interrupt service routine and the instance at which the need for the task or interrupt from an event arises). The different operations may take place at