Visual Walkthrough

is "an arrangement in which all its units assemble and work together according to a plan or program".

Example 1.1

A watch is a time-display system. Its components are its hardware, needles and battery. The beautiful dial, chassis and strap are also its components. All the components organise in the watch system in a certain specific way. The display system shows *time* of the day every second. The time showed continuously updates every second. The display system follows a *set of rules*. For example, one of the rules is that all needles move clockwise only. There are other rules also in time-display system.

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 in a computer hardware. The system is dedicated for either an application(s) or specific part of an application or product or a component of a large system."

The embedded systems have been defined in several ways in different books published. Given below is a series of definitions from different authors.

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 System 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 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".

1.1.3 Embedded Systems vs General Computing Systems

A computer is an example of general-purpose computing system. A computer is a system that has the following or more components.

sources, for example, Intel and Texas. ARM and Texas Instruments have developed the ARM families of the processors integrated with the DSP.

Example 2.7

ARM is used in embedded system design due to the following features.

1. The cores of ARM7, ARM9 and their DSP enhancements are available for embedding in systems. [Refer to *http://www.ti.com/sc/ docs/asic/modules/arm7.htm and arm9.htm*].
2. ARM9 enables design of setup boxes, cable modems, and wireless-devices such as mobile handsets. ARM9 has a single cycle 16 × 32 multiply accumulate unit. It operates at 200 MHz. It uses 0.15 µm GS30 CMOSs. It has a five-stage pipeline. It incorporates RISC. It integrates with a DSP when designing an ASIC (Application Specific Integrated Circuit) solution. An example is its integration with DSP is TMS320C55x from Texas. [Refer to *http://www.ti.com/sc/docs/asic/modules/arm7.htm and arm9.htm*]
3. ARM7 is a lower performance but very popular version of ARM. It operates at 80 MHz clock speed. It uses 0.18 µm based CMOSs. It has a three-stage pipeline. Using ARM7, a large number of embedded systems have recently become available.

8. Embedding a Multiprocessor or Dual Core using General-Purpose Processors (GPP)

An embedded system may require several processors or dual core processors. Real-time video processing and multimedia applications most often need a multiprocessor unit in the embedded system.

Example 2.8

Multiple ASIPs or dual-core processors are used in embedded system-design for implementing

1. **Real-time video and smart streaming graphic processors:** This is because the number of MAC operations needed per second may be more than what is possible from one DSP unit. An embedded system then may have to incorporate two or more processors running in synchronisation.
2. **High-definition television signal processors:** *High definition* means that the signals are processed for a noise-free, echo-cancelled transmission, and for obtaining a flat high-resolution image (1920 × 1020 pixels) on the television screen.
3. **Cell phone or digital camera:** These require suitably synchronised multiple processors. Number of tasks a mobile phone performs includes (a) speech signal-compression and coding, (b) dialing, (c) modulating and transmitting, (d) demodulating and receiving, (e) signal decoding and decompression, (f) keypad interface and display-interface handling, (g) Short Message Service (SMS) protocol-based messaging, and (h) SMS message display. A single processor does not suffice for completing all these tasks in the required time intervals.
3. **Video conferencing system:** A Quarter Common Intermediate Format (Quarter-CIF) is used in this system. Image pixel is just 144 × 176 as against 525 × 625 pixels in a video picture on TV. Even then, samples of the image have to be taken at a rate of 144 × 176 × 30 = 760320 pixels per second and have to be processed by compression before transmission on a telecommunication or Virtual Private Network (VPN). [*Note:* The number of frames should be 25 or 30 per second (as per the standard adopted) for real-time displays and in motion pictures]. A single DSP-based embedded system does not suffice to get real-time images during video conferencing.

Real-time operations require execution of algorithms fast and within strict deadlines. Multiple processors or dual core processors are used in this case. A single microprocessor does not meet the needs of the different tasks that have to be performed concurrently in real-time video processing and multimedia tasks. The operations of all the processors are synchronised in order to obtain an optimum

Simple approach with interesting examples

Example 8.4

An object-based model is used instead of ACVM sequential program and processes-based models. Figure 8.4 shows the features of classes objects, and inheritance interface in a model for an ACVM. The following can be the classes and objects.

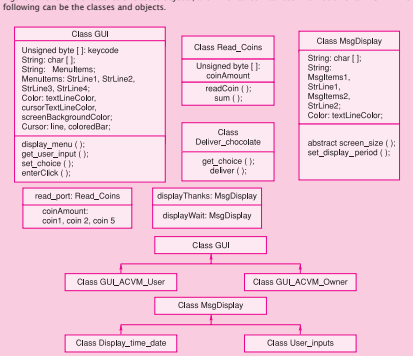

Fig. 8.4 Classes and objects and inheritance and interface features in a program model based for the ACVM

1. `Class GUI` for graphic-user interaction. It has two methods, `display_menu ( )` and `get_user_input ( )` for obtaining input for the choice of chocolate from the customer. It has the method `set_choice ( )` to set the choice selected.
2. `Class Read_Coins ( )` for reading the coins inserted. It has a method `readcoin ( ). readcoin ( )` reads one, two and five rupee coins from three ports and a method `sum ( )` for summing the total coins.
3. `Class Deliver_chocolate`. It has methods, `get_choice ( )` to get the choice and `deliver ( )` for delivering the chocolate.
4. `Class MsgDisplay`. It has methods `display_wait ( )` and `display_thanks ( )` for display wait message and thank message.

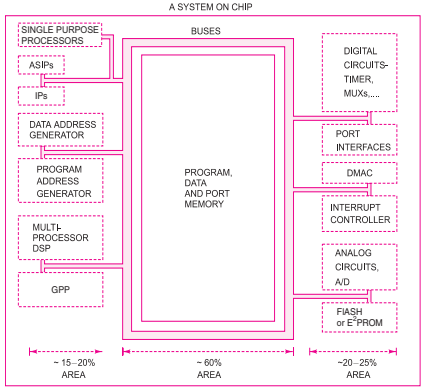

Fig. 2.2 SoC designed with system integration of software and communication processor, dual core processor including graphics processor, ASIP, IPs, program, data and port memory, and peripheral interfaces on a common bus

The SoC needs very large design efforts. The cost of development called Non-Recurring Engineering (NRE) cost is very high. Advantage of an SoC is that, the system has very high performance, functionalities and very low power dissipation. Therefore, SoCs are used in embedded systems, which are extensively and commonly used, need the small size, high system performance, much functionality, very low power dissipation and have low energy consumption. Examples of extensively used systems are mobile phones, tablets, personal computers, set-top boxes, digital TVs and cameras.

Example 2.1

An exemplary application of SoC is mobile phone. Components in SoC of a mobile phone are as follows:
1. Communication processor for 2G or 3G communication interface,
2. GPP 1.5 GHz dual-core processor including graphics processor (Apple, Android or Blackberry),
3. ASIP (Application Specific Instruction Processor) designed for video, audio and image processing,
4. Single-purpose processor designed for user interface for touch screen, and

3.3.6 External Memory Circuits

There are two sets of memory—program memory and data memory. Figure 3.5(a) shows how to interface the external program and data memory in 8051. Processor has two control signals PSEN and RD to control read from program memory or data memory. Processor has a control signal ALE to control use of AD0-AD7 as address or data at a given instance.

The 8051 has memory-mapped I/Os. It means memory and ports are assigned the addresses such that each have distinct range of addresses in the data-memory address space. Therefore, interfacing circuit design is identical to that for the memory when 8051 connects to the external ports and Parallel Peripheral Interface (PPI) chip Intel® 8255. External memory and ports are assigned the separate distinct addresses in 8051. Figure 3.5(b) shows the interface of 8051 and Intel® 8255, which has 24 port bits, PA.0 to PA.7, PB.0 to PB.7 and PC.0 to PC.7.

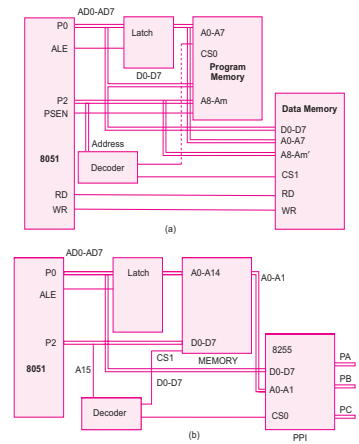

Fig. 3.5 (a) Interface circuit of external program and data memory and 8051
(b) Interface circuit of parallel peripheral interface Intel® 8255 and 8051

Self-explanatory figures to explain complex topics

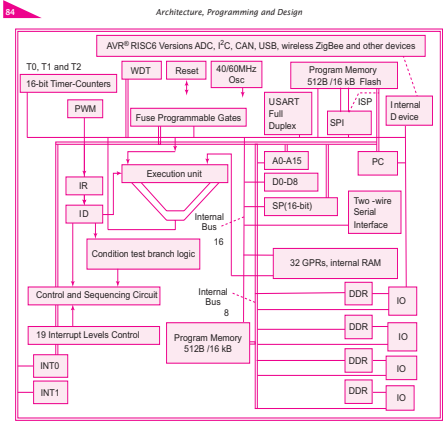

Fig. 3.9 ATMEL AVR® microcontroller architecture

1. AVR® is RISC Harvard memory architecture microcontroller. AVR® 8 consists of four ports with data direction register with each, ADC, USART, SPI, TWI, three timer-counters, two external interrupt pins, flash memory, EEPROM and SDRAM.
2. Six groups of AVR® versions are available. These possess for ADC, I²C, CAN, USB, wireless ZigBee and other devices.

3.5 ARM MICROCONTROLLERS

ARM® stands for Advance RISC Machines (microprocessors). ARM architecture offers high performance at very low power consumption. ARM Company designed ARM family of RISC superscalar processor architecture for VLSI implementation. The processor retains the best of CISC features also. The ARM"VLSIs are used widely as cores or chips. ARM MCUs (microcontrollers) are manufactured by Philips (now Nexperia), ST Microelectronics and Samsung. ARM MCUs consist of following hardware units:

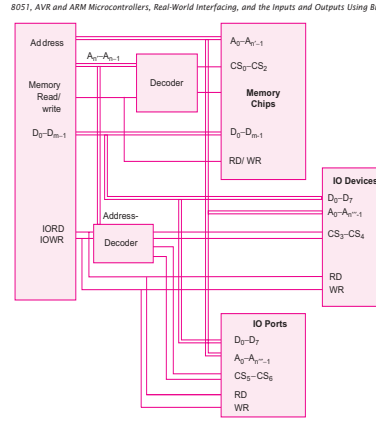

Fig. 3.14 I/O devices and components interfacing circuit with the ports

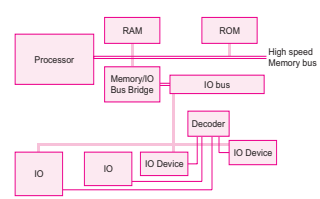

Fig. 3.15 I/O devices and components interfacing circuit using the I/O bus

186 *Architecture, Programming and Design*

SUMMARY

Following is a summary of the important points learnt in this chapter.

- I/O ports, I/O devices and timing devices are essential in any system. Two types of I/O ports and devices are serial and parallel. Serial communication is in synchronous (master-slave) mode or asynchronous mode and bits are transferred in successive time slots.
- A device connects and accesses from and to the system processor through either a parallel or serial I/O port. A device port may be full duplex or half duplex.
- A device or port has an assigned port address(es) using which the processor accesses the device or port-control register, status register and data. A device can use the handshaking signals before storing the bits at the port buffer or before accepting the bits from the port buffer.
- Serial communication bits are received at the receiver according to the clock phases of the transmitter. Synchronous serial communication bits from master carry the clock information also to the slave. Asynchronous serial communication bits from a device do not carry the clock information to the receiver and receiver clock phase is independent of the transmitter clock. However, the receiver clock adjusts its phase according to the received bits, for example, start bit.
- A popular asynchronous serial communication mode is UART. Bits are received at the receiver, independent of the clock phases at the UART (asynchronous serial input and output port) transmitter. UART in microcontrollers usually sends and receives a byte in a 10-bit or 11-bit format.
- Another popular asynchronous serial communication mode is RS232C, which is based on UART and is used to connect the data communication equipment such as modem with a data-terminal equipment such as computer.
- HDLC is a standard protocol for a synchronous communication data-link network between the devices.
- Widely used serial ports in the devices are SPI, SCI, SI and SDIO.
- Parallel communication is with or without handshaking signals. A number of embedded systems like parallel port or device interfaces to switches, keypad, encoders, motors, LCD controllers and touch screens. Special-purpose ports exist at the microcontroller for their interfacing. On-chip peripheral devices internally interface with the processor in the microcontrollers.
- Wireless communication is used for networking handheld devices over wireless personal area network.
- Timing and counting devices have a large number of uses in a system. A timer is essentially a counter getting the count-inputs (ticks) at regular time intervals.
- Use of buses simplifies the interfacing to multiple devices. Several devices can be placed on a common serial bus. Distributed networked architectures widely used serial buses are I^2C, CAN, USB and FireWire. Each device has an assigned device address. Using the device address of a receiver or slave, the master processor accesses the remote devices.
- I^2C bus is used between multiple ICs for inter-Integrated Circuit communication. A device, which initiates the communication and sends the clock pulses, is the master at an instance. A master can communicate to maximum 127 slaves.
- CAN bus is popularly used in centrally controlled network in automobile electronics.
- USB (Universal Serial Bus) is a standard for serial bus communication between the system and devices like scanner, keyboard, printer and mouse. There is a root-hub and all nodes have a tree-like structure.
- Very short distance devices interconnect to a PC or embedded system main bus through a common parallel bus, ISA, PCI or ARM. These buses connect to system memory bus through a bridge (switch).
- Internet-enabled embedded systems, network through protocols in TCP/IP protocol suite. Popularly used protocols are HTTP, TCP, UDP, IP and Ethernet.
- Embedded systems can interconnect and network without wires using IrDA, Bluetooth, 802.11 or ZigBee protocol compatible hardware and software support.

I/O Devices, Communication Buses and Distributed Networked Embedded Architectures 187

KEYWORDS

Asynchronous Communication	A communication in which a constant phase difference exists between transmitter and receiver bits and where the recovery clock is not maintained. The clocks that guide the transmitter and receiver are not synchronised. Time interval between which a set or frame of bytes transmits is not prefixed and is indeterminate. Asynchronous communication enables exchange of handshaking signals before and during the communication.
Bluetooth	A self-discovery and self-organizing network protocol for the wireless personal-area network and popularly used in mobile handheld devices.
CAN Bus	A standard bus used at the control-area network generally in automotive and industrial electronics.
COM Port	A port at the computer where a mouse, modem, serial printer or mobile smart-phone cradle connects for serial I/Os in UART mode and there are handshaking signals for exchange of signals before UART mode communication.
Control Register	A register for bits, which controls or programs the actions of a device. It is for a *write* operation only.
Control cum Status Register	A register at a port address that saves control and status bits and function as control register during write of commands and status register address during read of the status.
Counter	Unit for getting the count inputs on the occurrence of events that may be at irregular intervals.
Data Frame	A data frame consists of number of fields and each field has specified length. A sender sends a data frame at an instant and each receiver receives one frame. Then the receiver analyses each bit received in each field, and then it saves the data bits received at data field of the frame and undertakes necessary operations which may include sending an acknowledgement to the sender.
Debouncing	When a key is pressed, due to spring action, it vibrates and thus makes and breaks the contacts. This causes multiple 0s and 1s before the switch pressed state is accounted for. Debouncing by a hardware or software removes the signals due to bounces.
Delay	An action, communication, execution of codes or occurrence of an event blocked for a certain predefined period.
Demultiplexing	A way to separate a multiplexed input and direct the messages to multiple channels.
Device	A unit that has a processing element and that connects to the processor of embedded system internally or through the port or bus. It has fixed preassigned port addresses (device addresses) according to its interfacing or bus-controller circuit.
Device Decoder	A circuit to take the system address bus signals as the input and generate a device-select signal, CS, for the port address selection during the device read or write instructions of the system processor.
Event	A change of present condition, which gives an electric signal at a input or output pin or which changes a status bit or which interrupts the processor to enable some action.
Event Flag	A Boolean variable to indicate the event occurrence when it is true and can be a status register bit.
Field	A set of bits in a data frame; The frame consists of number of fields, before and after the field for data. Data field is compulsory and has a specified length. Each field has a specific purpose for the network transmitting and receiving devices or systems. A protocol specifies additional fields before data field and after data fields. Each field has specified number of bits. Each field has a specified sequence.

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

I/O Devices, Communication Buses and Distributed Networked Embedded Architectures 191

Review Questions

1. (a) What is the advantage of processor, which maps addresses of I/O ports and devices like a memory device? (b) Give a diagram to interface the port devices with the system buses.
2. Compare the advantages and disadvantages of data transfers using serial and parallel ports/devices.
3. (a) Explain three modes of serial communication, 'synchronous', 'isosynchronous' and 'asynchronous', from the serial devices with one example each (b) Describe and compare UART, RS232C and SDIO devices.
4. How do the following indicate the start and end of a byte or data frames? (a) UART (b) HDLC (c) CAN
5. What are the internal serial-communication devices in (a) 8051 and (b) 68HC11? Compare the modes of working of each of these.
6. A device port may have multibyte data input buffer(s) and data output buffer(s). What are the advantages of these?
7. What do you mean by a software timer (SWT)? How do the SWTs help in scheduling multiple tasks in real time?
8. Explain the advantages of Internet-enabled systems. How is the Internet-enabled device incorporated in the embedded system?
9. Explain the advantages of wireless devices. How does the wireless-devices network use different protocols?
10. What do you mean by the buses for networking of serial devices and the buses for networking of parallel devices?
11. Explain use of each control bit of (a) I^2C bus and (b) CAN bus. What is the advantage and disadvantage of negative acknowledgement?
12. How does the USB protocol function? Why does its features make it a widely used protocol?
13. What do you mean by a device attachment, configuration, reset, reconfiguration, bandwidth sharing (with other devices), and device detachment (while others are in operation) and reattachment?
14. What do you mean by plug and play devices? What are bus protocols of buses listed in Exercises given below that support plug and play devices?

Practice Exercises

1. How does the following device features help in an embedded system? (a) Schmitt trigger input (b) low voltage 5.3 V I/Os (c) Dynamically controlled impedance matching (c) PCS sub-unit, (d) PMA sub-unit, and (e) SerDes. Give one exemplary application of each.
2. PPP protocol for point-to-point networking has 8 starting flag bits, 8 address bits, 8 protocol specification bits, variable number of data bits, 16-bit CRC and 8 ending flag bits. Maximum number of bits per PPP frame can be 12064. How can many maximum number of bytes can be transferred per PPP frame? What is minimum percentage of overhead in the payload (frame)?
3. List the applications of a free-running counter, regularly interrupting timer and pulse-accumulator counter (PACT). How do you get PWM output from a PACT? How do you get DAC output from a PWM device?
4. A 16-bit counter is getting inputs from an internal clock of 12 MHz. There is a prescaling circuit, which prescales by a factor of 16. What are the time intervals at which overflow interrupts will occur from this timer? What will be period before which these interrupts must be serviced?
5. A new generation automobile has about 100 embedded systems. How do the bus arbitration bits, control bits for address and data length, data bits, CRC check bits, acknowledgement bits and ending bits in CAN bus help the networking devices distributed in an automobile embedded system.
6. Design a table that compares the maximum operational speeds and bus lengths and give two example of uses of each of the following serial devices: (a) UART (b) 1-wire CAN (c) Industrial I^2C (d) SM I^2C Bus (e) SPI of 68 Series Motorola Microcontrollers (f) Fault tolerant CAN (g) Standard Serial Port (h) MicroWire (i) I^2C

4. How to define the **software architecture** for software, extra functionalities and related systems and define decomposition of software into modules, components, appropriate protection strategies, and mapping of software
5. The coding for **implementation** of design using MUCOS and VxWorks RTOSes and use of the inter process communication (IPC) functions synchronizing and concurrent processing of task.
6. How state **machine concept** is used to model the design of a system

Only the functions implemented will be listed for the case studies. On-line contents (OLSs) associate with this book. OLCs give the details of system specifications, multiple tasks in the software architecture, synchronization models and exemplary coding steps and codes. Readers will find the details in answers of the practice exercises in the chapter.

13.1 CASE STUDY OF CODING FOR AN AUTOMATIC CHOCOLATE VENDING MACHINE USING MUCOS RTOS

Section 2.11.1 introduced the ACVM. The section described ACVM abstraction, hardware architecture, software architecture, extra functionalities required in the system, consideration of families of designs related to the system, and modular design and mapping. Following subsections describe how to use the software engineering approach in an ACVM design.

13.1.1 Requirements Study

Requirements of the machine can be understood through a requirement table given in Table 13.1.

Table 13.1 Requirements of an ACVM system

Requirement	*Description*
Purpose	To sell chocolate through an ACVM from which children can automatically purchase chocolates. The payment is by inserting the coins of appropriate amount into a coin-slot. [Adults are also welcome to use the machine!]
Inputs	1. Coins of different denominations through a coin slot 2. User commands
Signals, Events and Notifications	1. A mechanical system directs the coins to their appropriate port-`Port_1`, `Port_2` or `Port_5`. Each port generates an interrupt it receives a coin. An interrupt at `Port_1`, `Port_2` or `Port_5` gives a signal to the system that increases value of *amount-collected* by 1 or 2 or 5. 2. A selected menu choice gives a notification for an event in the system.
Outputs	1. An event so that the user gets a chocolate through a delivery port 2. A signal to run a command which subtracts the cost from the value of *amount-collected* 3. Displaying menus for GUIs, time and date, advertisements, and welcome messages
Functions	A child sends commands to the system for using a GUI (graphic user interface). GUIs consist of touch-screen displays and keypad units. The child inserts the coins for the cost of chocolate and the machine delivers the chocolate. If the coins are not inserted as per the cost of chocolate in reasonable times then all coins are refunded. If the coins are inserted of amount more than the cost of chocolate, the excess amount is refunded along with the chocolate. The coins for the chocolates purchased collect inside the machine in a collector channel, so that the owner can retrieve the money, again using appropriate commands to machine through the GUIs.

(Contd.)

3. `Task_Refund` waits for taking `SemFlag3` and then flushes to 0, the `SemAmtCount`, `SemMKey1`. It releases `SemFinish` on finishing the refund and accepting the `SemAmtCount` to make it 0. It sends `SemFinish` to `Task_ReadPorts`.
4. `Task_ExcessRefund` waits for taking `SemFlag2` and accepts `SemAmtCount` to decrease it by 8 and posts `SemFlag1` and releases `SemMKey1`. `Task_Display` waits for taking `SemFlag4`. It takes mutex, `SemMKey2` before passing the bytes to a stream for `Port_Display` and releases it after sending. It displays the mailbox messages at the message pointers, *Collect, *delivered, *refund, and *ExcessRefund.
5. Method `displayTimeDate ( )` displays at `Task_Display` gets a timeout notification through a mailbox message for time and date. The timeouts occur from `ISR_TimeDate` after every 1000 ticks of system clock. A timeout updates the time and date values at a pointer `*timeDate`. It posts into the mailbox `*timeDate` to `Task_Display` and `displayTimeDate ( )` uses it to display in the third line, right corner of the touch screen.

13.2 CASE STUDY OF DIGITAL CAMERA

Section 2.11.3 introduced the digital camera. The digital camera is an example of SoC [Section 2.1.1]. Section 2.11.3 listed the functions, hardware and software units and showed the hardware and software components in a simple digital camera. The following subsections describe the design steps of a digital camera and hardware and software architecture.

13.2.1 Requirements

Requirements of the digital camera can be understood through a requirement table given in Table 13.2.

Table 13.2 Requirements of a digital camera

Requirement	*Description*
Purpose	1. Digital recording and display of pictures 2. Processing to get the pictures of required brightness, contrast and color 3. Permanent saving of picture in file in a standard format at a magnetic stick 4. Transfer files to a computer through a port
Inputs	1. Intensity and color values for each picture horizontal and vertical row of pixels in a picture frame 2. Intensity and color values for unexposed (dark) area in each horizontal and vertical row of pixels 3. User-control inputs
Signals, Events and Notifications	1. User commands given as signals from switches/buttons
Outputs	1. Encoded file for a picture 2. Permanent store of the picture at a file on magnetic stick 3. Screen display of picture from the file after decoding 4. File output to an interfaced computer.
Functions of the System	1. A color LCD dot matrix displays the picture before shooting. This enables manual adjustment of view of the picture. 2. For shooting a shutter button is pressed. Then a charge-coupled device (CCD) array placed at the focus generates a byte stream in output after operations by ADC on analog output of each CCD cell.

(Contd.)

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

symphony (1808), one of the well known and most popular compositions of western classical music, which is often played in the orchestra using several musical instruments and conducted by a conductor [http://en.wikipedia.org/wiki/Symphony_No._5_(Beethoven)].

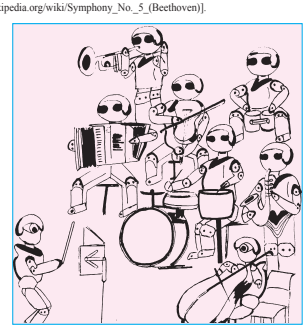

Fig. 13.18 Orchestra-playing robots

The present case study is to understand communication between the master (robot) and slaves (conductors). Commands and messages communicate between the master and slave.

Assume that there are k sensor inputs to the module. Also assume that q outputs generate to the actuators and the p outputs to **message boxes** A message box is called *mailbox* in certain OSes or *notification* in certain OSes are sequentially sent. The orchestrator is software which sequences, synchronizes the inputs from 1^{st} to k^{th} sensors and generates the messages and outputs for the actuators, display and message boxes at the specified instances and time intervals. Message boxes store the notifications, which initiate the tasks as per the notifications.

Figure 13.19(a) shows embedded software module Orchestrator-1 which runs at microcontroller 1. Figure 13.19(b) shows commands and messages communication between Orchestrator-x, Orchestrator-y and Orchestrator-z software modules at same or different microcontrollers.

A musical device communicates data to another using a protocol called MIDI (Musical Instrument Digital Interface). Most musical instruments are MIDI compatible and have MIDI IN and MIDI OUT connections, which are optically isolated with the musical instrument hardware. Three MIDI specifications define (i) what is a physical connector is, and (ii) what message format is used by connecting devices and controlling them in "real time" and standard for MIDI files. Each message consists of a command and corresponding data for that command. Data are sent in byte formats and are always between 0 and 127 and corresponding command bytes in a channel message are from 128 to 255.

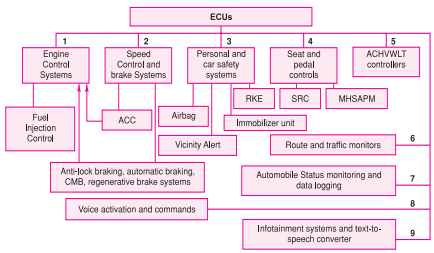

Fig. 13.25 Applications of the embedded systems in a car (RKE means Remote Key Entry and ACHVWLT means controllers to control the air-conditioning, heater, ventilation, windows, light and temperature)

9. Infotainment systems: Infotainment systems are as following: displayed text-to-speech converters, GPS based car location and surrounding area maps, cached traffic reports for real-time traffic monitoring, cassette player, car phones, audio CD player, LCD screen, touch panel screen, satellite or Internet Radio, VCD/DVD players.

Hardware of car ECUs can be designed using ASICs and microcontrollers and DSPs, for example, 80x51, 68HC11/12, PIC, C167, ADSP2106x, 68HC0x, MCORE, Star12, TMS470, Hitachi H8S2xxx series, and ARM 9 based ST9 series. Section 13.6 describes a case study of software implementation aspects of an ACC system in a car.

13.6 CASE STUDY OF AN EMBEDDED SYSTEM FOR AN ADAPTIVE CRUISE CONTROL (ACC) SYSTEM IN A CAR

The choice of case study of an ACC is taken up to understand a control system design and also to understand use of the RTOS for code-implementation. The system has number of ports for data inputs and outputs. The system uses a control algorithm. Sections 13.6.1 and 13.6.2 give the design steps of requirements and class diagram of an ACC system tasks. Sections 13.6.3 and 13.6.4 describe hardware and software architecture. Section 13.6.5 describes ACC tasks synchronization model.

13.6.1 Requirements

Requirements of the ACC system can be understood through a requirement table given in Table 13.6.

MISRA-C is a standard for C language software and defines the guidelines for automotive systems for using C. MISRA-C version 2 (2004) specified 141 rules for coding and gave a new structure for C. Details can be found at http://www.misra.org.uk. Figure 13.31 shows important rules and coding standards in MISRA-C.

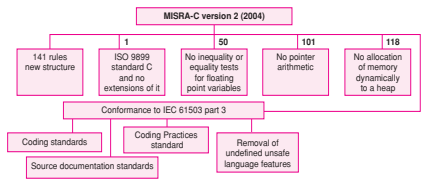

Fig. 13.31 Important rules and coding standards in MISRA-C

A few rules are discussed next. Its *first rule* is that all C codes used in an automobile must conform to ISO 9899 standard C and no extensions of it should be permitted. Rule 43 does not permit use of implicit cast which may result in a loss of information. Rule 50 does not permit inequality or equality tests for floating-point variables. Floating-point calculations undergo rounding-off errors. The logic for introducing this rule is as follows: Consider 'If ((1/3) * 3 = = 1) then ….". 1/3 = 0.33333 with 3 in the last digit or 4 on the last digit and the result is always uncertain. Rule 65 does not permit use of a floating-point number as a loop counter. Rule 101 does not permit use of pointer arithmetic. It is similar to the rule in Java. Rule 118 does not permit allocation of memory dynamically to a heap. Dynamic allocation has risk of additional memory allocation than available in the system, which may cause memory leaks.

13.8 CASE STUDY OF AN EMBEDDED SYSTEM FOR A SMART CARD, ACCESS CONTROL SYSTEMS (SMART CARDS, RFIDS, FINGERSCAN)

Section 13.8.1 gives the requirements and functioning of the smart-card communication system. Section 13.8.2 gives the class diagram. Sections 13.8.3 and 13.8.4 give the hardware and software architecture and synchronization model. Section 13.8.5 gives the exemplary codes.

13.8.1 Requirements

Assume a contact-less smart card for bank transactions. Let it not be magnetic. [The earlier card used a magnetic strip to hold the non-volatile memory. Nowadays, it is EEPROM that is used to hold non-volatile application data.] Requirements of the smart-card communication system with a host can be understood through a requirement table given in Table 13.7.

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

and receives encrypted string "*Closure Permitted*". Tasks delete on deciphering.

4. `task_PW` after encryption on taking the pending `SemPW` is to send the string *requestPW*. When it takes `SemPW`, it sends the *requestPW* into the `MsgQPW`. `task_ReadPort` will send it to the host through the IO Port, `Port_IO`, in order to identify the user at the host.
5. `task_Appl` runs on taking the semaphore `SemAppl` and executes the operations. The operation may (i) modify user password, (ii) print a mini statement of the bank account of the user, (iii) eject requisite cash from the host, (iv) request for accepting the envelope with cash, (v) request for a print of this transaction, and (vi) request for a transfer to another party. It interacts through `task_ReadPort` by sending the messages through the queue `MsgQAppl`.

The task synchronization model is also shown in Fig.13.34.

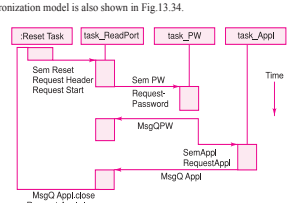

Fig. 13.34 Smart-card tasks and their synchronization model

13.9 CASE STUDY OF A MOBILE-PHONE SOFTWARE FOR KEY INPUTS

Mobile phones are smart. Each has many APIs. Examples 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 shows, and camera.

Mobile phones with large touch screens use a virtual keypad. Mobile phones with small screens use a T9 keypad. Blackberry provides a Qwerty keyboard. The present case study relates to 'SMS create application' in a mobile phone.

Section 13.9.1 gives the requirements of the SMS create and send application. Section 13.9.2 gives the classes and class diagrams. Section 13.9.3 gives the state diagram, and Section 13.9.4 gives communication hardware. Section 13.9.5 describes software architecture.

13.9.1 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 an application. Persistence memory addresses are used such that as soon as a change is made in the byte, it persists even after the power switches off. Further, when there

6.10 CONTEXT AND THE PERIODS FOR CONTEXT SWITCHING

Process of change of running program at the CPU to a new program is as follows:

1. Save the address (instruction pointer) from where the program will begin on return and save processor status word,
2. Save current program's registers, and other program parameters,
3. Find the address (instruction pointer) from where a new program begins,
4. Load the program's address into instruction pointer (program counter),
5. Load the new program's status word, registers, and other program parameters, and
6. Execute instructions of the new program. [Program means foreground program, process, thread, task, routine, ISR, signal handler or exception handler.]

Context of a program means, the address (instruction pointer) from where the program will begin on return, processor-status word, current program's registers, and other program parameters. Figure 6.20(a) shows current program context.

Steps 1 and 2 mean, saving the currently running program context. Steps 4 and 5 mean loading the new program context. Figure 6.20(b) shows steps on context switching when new program executes with new context.

Context saving is essential. The process ensures that (i) on return the saved program starts from same state as at the instance of change to new program, (ii) when new program starts then it also starts from same state as at the instance of earlier change from the program.

Context switching is performed in the system when

1. A foreground program interrupts and ISR starts execution,
2. When an ISR interrupts by higher priority ISR and new ISR starts,
3. When returning to previously running program,
4. When a signal is issued and signal handler executes,
5. When an exception is thrown on exceptional condition and exception handler (catch function) executes, and
6. When a thread (or task or process) starts waiting for a message or parameter and blocks, and system software starts new thread.

Figure 6.20(c) shows context switching to new routine and another switch on return to current routine. Figure 6.20(d) shows context switching for a new routine and another switch on higher priority routine.

Context switching period equals the processor time spent in saving the context plus time taken in loading the new context.

Each running program has a context at an instant. Context reflects a CPU state [instruction pointer, stack pointer(s), registers and program state (variables that should not be modified by another routine)]. Context saving on the call of another program is essential before switching to another context. Context loading is essential so that a new one starts from the previously left context. Program means foreground program, process, thread, task, routine, ISR, signal handler or exception handler.

6.11 INTERRUPT LATENCY

When a processor interrupts the service of the interrupt by execution, the ISR may not start immediately after context switching. The interval between occurrence of interrupt and start of execution of the ISR is called interrupt latency.

10.1 OPERATING SYSTEM SERVICES

10.1.1 OS Services Goal

OS services Goal of perfection and correctness'. OS facilitates the following:

1. **Easy sharing of resources as per schedule and allocations**. Resources mean processor(s), memory, I/O, devices, pipes, sockets, system timer, keyboard, displays, printer and other such resources, which processes (tasks or threads) request from the OS. No processing task or thread uses any resource until it has been allocated by the OS at a given instance.
2. **Easy implementation** of application software with the given system hardware. An application uses the OS functions and processes which are provided in the OS.
3. **Scheduling,** context switching and interrupt-servicing mechanisms.
4. **Management of the processes**, tasks, threads, memory, IPCs, devices, and other functions. [Management means creation, resources allocation, resources freeing, scheduling or synchronising, and deletion.]
5. Files, I/O and Network **subsystems and protocols**.
6. **Portability** of the application on different hardware configurations.
7. **Interoperability** of the application on different networks.
8. **Common set of interfaces** that integrates various devices and applications through standard and open systems.
9. Easy use of the interfacing functions, **GUIs and APIs**.
10. Maximising the **system performance** to let different processes (or tasks or threads) share the resources most efficiently. OS provides the protection and security. Examples of security breach are tasks as follows: obtaining illegal access to other task data directly without system calls, overflow of stack areas into the memory, and overlaying of process and control blocks and thread stacks in memory.

10.1.2 User and Supervisory Mode Structure

When using an OS, the processor in the system runs in two modes. There is a clock, called system clock. At every clock tick of system-clock, there is an interrupt. On interrupt, the system time updates, the system context switches to supervisory mode from the user mode. After completing the supervisory functions in the OS, the system context switches back to user mode.

1. User Mode

User function call, which is not a system call, is not permitted to read and write into the protected memory allotted to the OS functions, data, stack and heap. That protected memory space is also called kernel space.

2. Supervisory Mode

The OS runs the privileged functions and instructions in protected mode and the OS (more specifically, the kernel) only accesses the hardware resources and protected area memory. [The term *kernel* means nucleus.] Kernel codes run in protected mode. Only a system call is permitted to read and write into the protected memory allotted to the OS functions, data, stack and heap.

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

Table 8.2 UML basic elements

Modeling Diagram	*What does it Model and Show?*	*Exemplary Diagrammatic Representation*
Object	An instance of a class that is a functional entity formed by copying the states, attributes and behavior from a class.	Rectangular box with object identity followed by semicolon and class identity [Figure 8.15(d)]
Active Object	An active class defines an active object instance of an active class. A process or thread is equivalent to active object in UML, because active object posts the signals like thread and can wait before start 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, behaviors and behaviors for the signals. Active class in addition, defines the control by signal behaviors (for a signaling object, which can be posted and for which it may wait before start or resuming). Thus, there is control on the class behavior.	Rectangular box with thick border lines and inner divisions for the class names for the identity, attributes and behaviors (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 wait for start or resumption. Signal-object behavior defines in behavior (operation method) for the interprocess communication. [Signal is software instruction or method (function), which generates interrupt.] Signal object has attributes (parameters that 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].
Stereotype	An unpacked collection of elements (attributes or behaviors) 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 [Figure 8.15(c)]
Anonymous Object	An object without identity	Rectangular box with no object identity before the semicolon and class identity [Figure 8.15(e)]
Package	A packed collection of classes and objects.	A rectangular box with inner boxes for each class with name for class identity. Package name is given over the top of the box [Figure 8.15(b)].
State	A state which undergoes state transitions and which may depend on previous state.	Rounded Rectangle with state name for its identity and with an arrow from the box. The arrow indicates a transition [Figure 8.15(f)].

A conceptual design modeling can follow UML approach. Figure 8.16 shows UML diagrams. A conceptual design can use the 'User Diagram', 'Object Diagram', 'Sequence Diagram', 'State Diagram', 'Class Diagram' and 'Activity Diagram'.