A device driver accesses a parallel or serial port, keyboard, mice, disk, network, display, file, pipe and socket at specific addresses. An OS also provides device driver codes for system-port addresses and for hardware access mechanisms.

A device manager software provide codes for detecting the presence of devices, for initializing these and for testing the devices that are present. The manager includes software for allocating and registering port (in fact, it may be a register or memory) addresses for the various devices at distinctly different addresses, including codes for detecting any collision between these, if any. It ensures that any device accesses to one task only at any given instant. It takes into account that virtual devices may also have addresses that are allocated by the manager.

An OS also provides and executes modules for managing devices that associate with an embedded system. The underlying principle is that at an instant, only one physical or virtual device should get access to or from one task only.

Sections 4.2.4 and 8.6.1 will describe device drivers and device management in detail. The OS also provides and manages virtual devices such as pipes and sockets. Sections 7.14 and 7.15 describe these in detail.

For designing embedded-software, two types of devices are considered: physical and virtual. Physical devices include keypad, printer or display unit. A virtual device could be a file or pipe or socket or RAM disk. Device drivers and device manager software are needed in the system. The RTOS includes device-drivers and a device manager to control and facilitates the use of the number of physical and virtual devices in the system.

1.4.8 Software Tools for Designing an Embedded System

Table 1.2 lists the applications of software tools for assembly language programming, high level language programming, RTOS, debugging and system integration.

Table 1.2 Software modules and tools for designing of an embedded system

Software Tools	*Application*
Editor	For writing C codes or assembly mnemonics using the keyboard of the PC for entering the program. Allows the entry, addition, deletion, insert, appending previously written lines or files, merging record and files at the specific positions. Creates a source file that stores the edited file. It also has an appropriate name [provided by the programmer].
Interpreter	For expression-by-expression (line-by-line) translation to machine-executable codes.
Compiler	Uses the complete set of codes. It may also include codes, functions and expressions from the library routines. It creates a file called object file.
Assembler	For translating assembly mnemonics into binary opcodes (instructions), that is, into an executable file called binary file and for making a list file that can be printed. The list file has address, source code (assembly language mnemonics) and hexadecimal object codes. The file has addresses that reallocate during the actual run of the assembly language program.

(Contd)

Software Tools	*Application*
Cross assembler	For converting object codes or executable codes for a processor to other codes for another processor and vice versa. The cross-assembler assembles the assembly codes of the target processor as the assembly codes of the processor of the PC used in system development. Later, it provides the object codes for the target processor. These codes will be the ones actually needed in the final developed system.
Simulator	To simulate all functions of an embedded system circuit including that or additional memory and peripherals. It is independent of a particular target system. It also simulates the processes that will execute when the codes of a particular processor execute.
Source-code engineering software	For source code comprehension, navigation and browsing, editing, debugging, configuring (disabling and enabling the C++ features) and compiling.
RTOS	Refer Chapters 8 to 10.
Stethoscope	For dynamically tracking the changes in any program variable or parameter. It demonstrates the sequence of multiple processes (tasks, threads, service routines) that execute and also records the entire time history.
Trace scope	To help in tracing the changes in modules and tasks with time on the X-axis. A list of actions also produces the desired time scales and the time expected to be taken for different tasks.
Integrated development environment	This is a development software and hardware environment that consists of simulators with editors, compilers, assemblers, RTOS, debuggers, stethoscope, tracer, emulators, logic analyzers, and application code burners in PROM or flash.
Prototyper	This simulates and does source code engineering including compiling, debugging and, browsing and summarizing the complete status of the final target system during the development phase.
Locator#	This uses a cross-assembler output and a memory allocation map and provides the locator program output as a hex-file. It is the final step of the software design process for an embedded system.

The locator program output is in the Intel hex file or Motorola S-record format.

Software tools are used to develop software for designing an embedded system. Debugging tools, such as a stethoscope, trace scope, and sophisticated tools such as an integrated development environment and prototype development tools, are needed for the integrated development of system software and hardware.

1.4.9 Software Tools Required in Exemplary Cases

Table 1.3 gives the various tools needed to design exemplary systems.

RTOS is essential in most embedded systems to process multiple tasks and ISRs. Embedded systems for medium scale and sophisticated applications need a number of sophisticated software and debugging tools.

Table 1.3 Software tools required in exemplary systems

Software Tools	*Automatic Chocolate Vending Machine*[&]	*Data Acquisition System*	*Robot*	*Mobile Phone*	*Adaptive Cruise Control System with String Stability*[#]	*Voice Processor*
Editor	Yes	Yes	Yes	Yes	Yes	NR
Interpreter	Yes	NR	Yes	NR	NR	NR
Compiler	Yes	Yes	Yes	Yes	Yes	Yes
Assembler	Yes	Yes	Yes	No	No	No
Cross Assembler	NR	Yes	Yes	No	No	No
Locator	Yes	Yes	Yes	Yes	Yes	Yes
Simulator	NR	Yes	Yes	Yes	Yes	Yes
Source code engineering software	NR	NR	NR	Yes	Yes	Yes
RTOS	Yes	MR	Yes	Yes	Yes	Yes
Stethoscope	NR	NR	NR	Yes	Yes	Yes
Trace scope	NR	NR	NR	Yes	Yes	Yes
Integrated development environment	NR	Yes	Yes	Yes	Yes	Yes
Prototyper	NR	No	No	Yes	Yes	Yes

Note: NR means not required. MR means may be required in a specific complex system but not compulsorily needed.

1.5 EXAMPLES OF EMBEDDED SYSTEMS

Embedded systems have very diversified applications. A few select application areas of embedded systems are telecommunications, smart cards, missiles and satellites, computer networking, digital consumer electronics, and automotives. Figure 1.9 shows the applications of embedded systems in these areas.

A few examples of *small scale embedded system* applications are as follows:

1. Point of sales terminals: automatic chocolate vending machine
2. Stepper motor controllers for a robotics system
3. Washing or cooking systems
4. Multitasking toys
5. Microcontroller-based single or multidisplay digital panel meter for voltage, current, resistance and frequency
6. Keyboard controller
7. SD, MMI and network access cards
8. CD drive or hard disk drive controller

9. The peripheral controllers of a computer, for example, a CRT display controller, a keyboard controller, a DRAM controller, a DMA controller, a printer controller, a laser printer controller, a LAN controller, a disk drive controller
10. Fax or photocopy or printer or scanner machine
11. Remote (controller) of TV
12. Telephone with memory, display and other sophisticated features

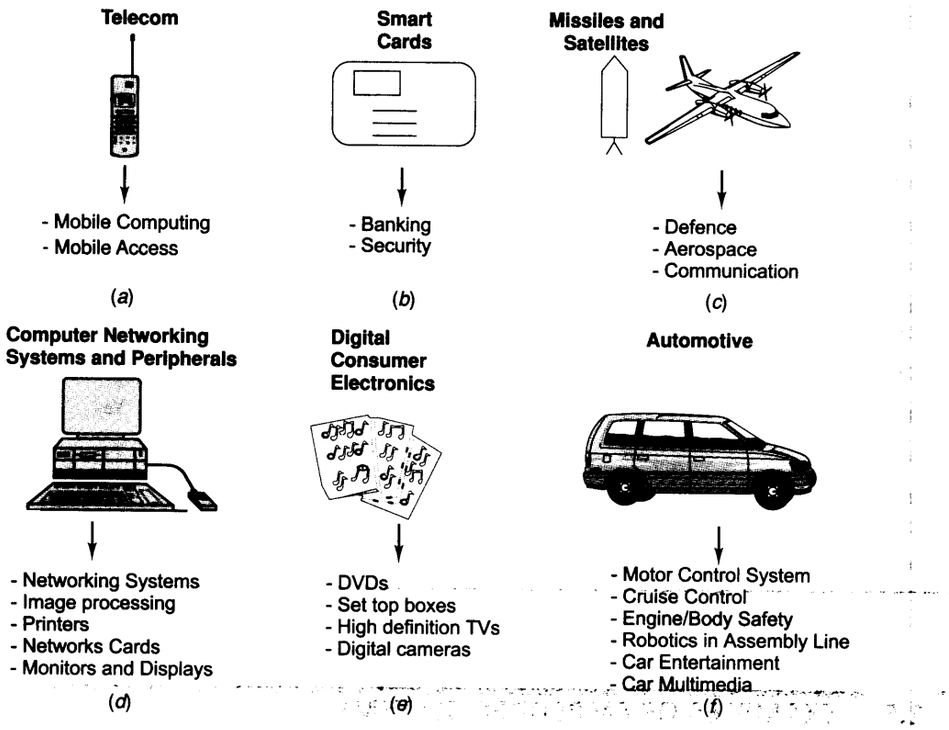


Fig. 1.9 Applications of the embedded systems in various areas

13. Motor controls systems—for example, an accurate control of speed and position of the d.c. motor, robot and CNC machine; automotive applications such as closed loop engine control, dynamic ride control, and an antilock braking system monitor
14. Electronic data acquisition and supervisory control system
15. Electronic instruments, such as an industrial process controller
16. Electronic smart weight display system and an industrial moisture recorder cum controller
17. Digital storage system for a signal wave form or for electric or water meter reading system
18. Spectrum analyzer
19. Biomedical systems such as an ECG LCD display cum recorder, a blood-cell recorder cum analyzer, and a patient monitor system

Some examples of *medium scale embedded systems* are as follows:

20. Computer networking systems, for example, a router, a front-end processor in a server, a switch, a bridge, a hub and a gateway
21. For Internet appliances, there are numerous application systems (i) An intelligent operation, administration and maintenance router (IOAMR) in a distributed network and (ii) Mail client card to store e-mail and personal addresses and to smartly connect to a modem or server
22. Entertainment systems such as a video game and a music system
23. Banking systems, for example, bank ATM and credit card transactions
24. Signal tracking systems, for example, an automatic signal tracker and a target tracker
25. Communication systems such as a mobile communication SIM card, a numeric pager, a cellular phone, a cable TV terminal and a FAX transceiver with or without a graphic accelerator
26. Image filtering, image processing, pattern recognizer, speech processing and video processing
27. Video games
28. A system that connects a pocket PC to the automobile driver mobile phone and a wireless receiver. The system then connects to a remote server for Internet or e-mail or to a remote computer at an ASP (application service provider)
29. A personal information manager using frame buffers in handheld devices
30. Thin client [A thin client provides disk-less nodes with remote boot capability]. Application of thin-client accesses to a data centre from a number of nodes; in an Internet laboratory accesses to the Internet leased line through a remote server.
31. Embedded firewall / router using ARM7/ multiprocessor with two Ethernet interfaces and interfaces support to PPP, TCP/IP and UDP protocols.

Examples of *sophisticated embedded systems* are as follows:

32. Mobile smart phones and computing systems
33. Mobile computer
34. Embedded systems for wireless LAN and for convergent technology devices
35. Embedded systems for video, interactive video, broadband IPv6 (Internet Protocol version 6) Internet and other products, real time video and speech or multimedia processing systems
36. Embedded interface and networking systems using high speed (400 MHz plus), ultra high speed (10 Gbps) and a large bandwidth: Routers, LANs, switches and gateways, SANs (Storage Area Networks), WANs (Wide Area Networks)
37. Security products and high-speed Network security. Gigabit rate encryption rate products

1.6 EMBEDDED SYSTEM-ON-CHIP (SoC) AND USE OF VLSI CIRCUIT DESIGN TECHNOLOGY

Lately, embedded systems are being designed on a single silicon chip, called *System on chip (SoC), a design innovation*. SoC is a system on a VLSI chip that has all the necessary analog as well as digital circuits, processors and software.

A SoC may be embedded with the following components:

1. Embedded processor GPP or ASIP core,
2. Single purpose processing cores or multiple processors,
3. A network bus protocol core,
4. An encryption function unit,

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.

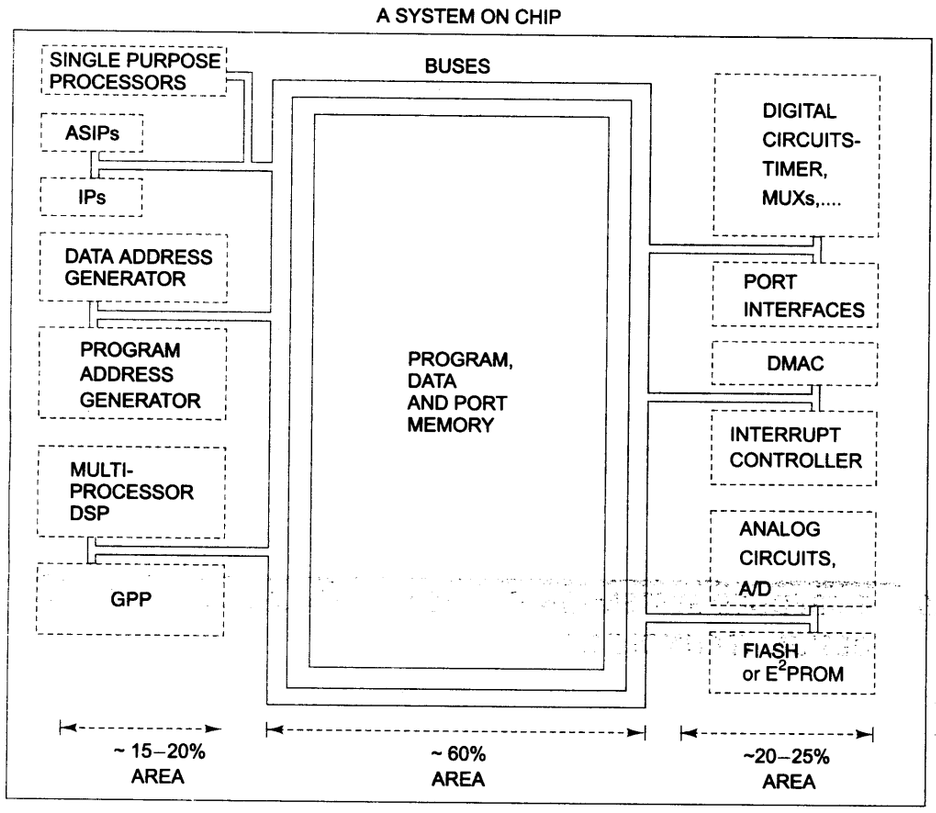


Fig. 1.10 A SoC embedded system and its common bus with internal ASIPs, internal processors, IPs, shared memories and peripheral interfaces

1.6.1 Application Specific IC (ASIC)

ASICs are designed using the VLSI design tools with the processor GPP or ASIP and analog circuits embedded into the design. The designing is done using the Electronic Design Automation (EDA) tool. [For design of an ASIC, a High-level Design Language (HDL) is used].

1.6.2 IP Core

On a VLSI chip, there may be integration of high-level components. These components possess gate-level sophistication in circuits above that of the counter, register, multiplier, floating point operation unit and ALU. A standard source solution for synthesizing a higher-level component by configuring an FPGA core or a core of VLSI circuit may be available as an Intellectual Property, called (IP). The designer or the designing company holds the copyright for the synthesized design of a higher-level component for gate-level implementation of an IP. One might have to pay royalty for every chip shipped. An embedded system may incorporate several IPs.

- An IP may provide hardwired implementable design of a *transform*, an *encryption algorithm* or a *deciphering algorithm.*
- An IP may provide a design for *adaptive filtering* of a signal.
- An IP may provide a design for implementing Hyper Text Transfer Protocol (HTTP) or File Transfer Protocol (FTP) or Bluetooth protocol to transmit a web page or a file on the Internet.
- An IP may be designed for a USB or PCI bus controller. [Sections 3.10.3 and 3.12.2]

1.6.3 FPGA Core with Single or Multiple Processors

Suppose an embedded system is designed with a view to enhancing functionalities in future. An FPGA core is then used in the circuits. It consists of a large number of programmable gates on a VLSI chip. There is a set of gates in each FPGA cell, called macro cell. Each cell has several inputs and outputs. All cells interconnect like an array (matrix). Each interconnection is programmable through the associated RAM in an FPGA programming tool. An FPGA core can be used with a single or multiple processor.

Consider the algorithms for the following: Fourier transform (FT) and its inverse (IFT), DFT or Laplace transform and its inverse, compression or decompression, encrypting or deciphering, specific pattern recognition (for recognizing a signature or finger print or DNA sequence). We can configure an algorithm into the logic gates of FPGA. It gives hardwired implementation for a processing unit. It is specific to the needs of the embedded system. An algorithm of the embedded software can implement in one of the FPGA sections and another algorithm in its other section.

FPGA cores with a single or multiple processor units on chip are used. One example of such core is Xilinx Virtex-II Pro FPGA XC2VP125. XC2VP125 from Xilinx has 125136 logic cells in the FPGA core with four IBM PowerPCs. It has been used as a data security solution with encryption engine and data rate of 1.5 Gbps. Other examples of embedded systems integrated with logic FPGA arrays are DSP-enabled, real-time video processing systems and line echo eliminators for the Public Switched Telecommunication Networks (PSTN) and packet switched networks. [A packet is a unit of a message or a flowing data such that it can follow a programmable route among the number of optional open routes available at an instance.]

1.7 COMPLEX SYSTEMS DESIGN AND PROCESSORS

1.7.1 Embedding a Microprocessor

A General Purpose Processor microprocessor can be embedded on a VSLI chip. Table 1.4 lists different streams of microprocessors embedded in a complex system design.

Table 1.4 Important microprocessors used in embedded systems

Stream	*Microprocessor Family*	*Source*	*CISC or RISC or Both features*
Stream 1	68HCxxx	Motorola	CISC
Stream 2	80x86	Intel	CISC
Stream 3	SPARC	Sun	RISC
Stream 4	ARM	ARM	RISC with CISC functionality

1.7.2 Embedding a Microcontroller

Microcontroller VLSI cores or chips for embedded systems are usually among the five streams of families given in Table 1.5.

Table 1.5 Major microcontrollers[@] used in the embedded systems

Stream	*Microcontroller Family*	*Source*	*CISC or RISC or Both*
Stream 1	68HC11xx, HC12xx, HC16xx	Motorola	CISC
Stream 2	8051, 8051MX	Intel, Philips	CISC
Stream 3	PIC 16F84 or 16C76, 16F876 and PIC18	Microchip	CISC
Stream 4*	Microcontroller Enhancements of CORTEX-M3 ARM9/ARM7 from Philips, Samsung and ST Microelectronics	ARM, Texas, Philips, Samsung and ST Microelectronics etc.	RISC Core with CISC functionality

[@] Other popular microcontrollers are as follows. (i) Hitachi H8x family and SuperH 7xxx. (ii) Mitsubishi 740, 7700, M16C and M32C families. (iii) National Semiconductor COP8 and CR16 /16C. (iv) Toshiba TLCS 900S (v) Texas Instruments MSP 430 for low voltage battery based system. (vi) Samsung SAM8. (vii) Ziglog Z80 and eZ80

1.7.3 Embedding a DSP

A *digital signal processor (DSP)* is a processor core or chip for the applications that process digital signals. [For example, filtering, noise cancellation, echo elimination, compression and encryption applications.] Just as a microprocessor is the most essential unit of a computing system, a DSP is essential unit of an embedded

system in a large number of applications needing processing of signals. Exemplary applications are in image processing, multimedia, audio, video, HDTV, DSP modem and telecommunication processing systems. DSPs also find use in systems for recognizing image pattern or DNA sequence.

DSP as an ASIP is a single chip or core in a VLSI unit. It includes the computational capabilities of a microprocessor and Multiply and Accumulate (MAC) units. A typical MAC has a 16 × 32 MAC unit.

DSP executes discrete-time, signal-processing instructions. It has Very Large Instruction Word (VLIW) processing capabilities; it processes Single Instruction Multiple Data (SIMD) instructions; it processes Discrete Cosine Transformations (DCT) and inverse DCT (IDCT) functions. The latter are used in algorithms for signal analyzing, coding, filtering, noise cancellation, echo elimination, compressing and decompressing, etc.

Major DSPs for embedded systems are from the three streams given in Table 1.6.

Table 1.6 Important digital signal processor@ used in the embedded systems

Stream	*DSP Family*	*Source*
Stream 1	TMS320Cxx, OMAP[1]	Texas
Stream 2	Tiger SHARC	Analog Device
Stream 3	5600xx	Motorola
Stream 4	PNX 1300, 1500[2]	Philips

[1]For example, TMS320C62XX a fixed point 200 MHz DSP (Section 2.3.5).
[2]Media processor, which besides multimedia DSP operations, also does network stream data packet processing.

1.7.4 Embedding an RISC

A RISC microprocessor provides the speedy processing of instructions, each in a single clock-cycle. This facilitates pipelining and superscalar processing. Besides greatly enhanced capabilities mentioned above, there is great enhancement of speed by which an instruction from a set is processed. Thumb® instruction set is a new industry standard that also gives a reduced code density in ARM RISC processor. RISCs are used when the system needs to perform intensive computation, for example, in a speech processing system.

1.7.5 Embedding an ASIP

ASIP is a processor with an instruction set designed for specific application areas on a VLSI chip or core. ASIP examples are microcontroller, DSP, IO, media, network or other domain-specific processor.

Using VLSI design tools, an ASIP with instructions sets required in the specific application areas can be designed. The ASIP is programmed using the instructions of the following functions: DSP, control signals processing, discrete cosine transformations, adaptive filtering and communication protocol-implementing functions.

1.7.6 Embedding a Multiprocessor or Dual Core Using GPPs

In an embedded system, several processors or dual core processors may be needed to execute an algorithm fast within a strict deadline. For example, in real-time video processing, the number of MAC operations needed per second may be more than is possible from one DSP unit. An embedded system then incorporates two or more processors running in synchronization. An example of using multiple ASIPs is high-definition television signals processing. [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.] A cell phone or digital camera is another application with multiple ASIPs.

In a cell phone, a number of tasks have to be performed: (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 (h) SMS message display. For all these tasks, a single processor does not suffice. Suitably synchronized multiple processors are used.

Consider a video conferencing system. In this system, a quarter common intermediate format—Quarter-CIF—is used. The number of image pixels 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 are 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 video processing and multimedia applications most often need a multiprocessor unit in the embedded system.

Multiple processors or dual core processors are used when a single microprocessor does not meet the needs of the different tasks that execute concurrently. The operations of all the processors are synchronized to obtain optimum performance.

1.7.7 Embedded Processor/Embedded Microcontroller

An embedded processor is a processor with special features that allow it to embed multiple processes into the system.

Real time image processing and aerodynamics are two areas where fast, precise and intensive calculations and fast context switching (from one program to another) are essential. Embedded processor is the term sometimes used for processor that has been a specially designed such that it has the following capabilities:

1. Fast context switching and thus lower latencies of the tasks in complex real time applications. [Section 4.6] Fast context switching means that the calling program or interrupted service routine CPU registers save and retrieve fast [Section 4.6].
2. 32-bit or 64-bit atomic addition and multiplication, and no shared data problem in the operations with large operands with each operand placed in two or four registers. [Section 7.8.1]
3. 32-bit RISC core for fast, more precise and intensive calculations by the embedded software.

Embedded microcontroller is the term sometimes used for specially designed microcontrollers that have the following capabilities:

1. When a microcontroller has internal RAM, large flash or ROM, timer, interrupt handler, devices and peripherals and there is no external memory or device or peripheral required for the given application.
2. Fast context switching and thus lower latencies of the tasks in complex real time applications. For example, ARM and 68HC1x microcontrollers save all CPU registers fast

An embedded processor is term used for processors with fast processing, fast context-switching and atomic ALU operations. An embedded microcontroller is the term used for a microcontroller that has internal RAM, large flash or ROM, timer, interrupt handler, internal devices and internal peripherals and there is no external memory or device or peripheral required for the given application.

Complex System Embedded Processors Table 1.7 gives different processors that can embed in a complex system.

Table 1.7 Processors in complex embedded systems

Processor	*Application*	*Advantage*	*Disadvantage*
General Purpose Microprocessor	When intensive computations are required, caches are used and pipeline and superscalar operations are needed and large embedded software is to be located in the external memory cores or chips.	No engineering cost for designing the processor.	Additional redundant execution units that are not needed in the given system design
Microcontroller	Used with internal memory, devices and peripherals and when embedded software is to be located in the internal ROM or flash.	No engineering cost for designing the processor with internal memory, devices and peripherals.	Additional manufacturing costs and redundant application units which are not needed in the given system design.
DSP	Used with signal processing-related instructions for filters, image, audio, and video and CODEC operations.	No engineering cost involved for designing the signal processor.	Manufacturing cost may be high.
Single purpose processors and application specific system processor	Control IO and bus operations and peripherals and devices.	They support other processing units in the system and execute specific hardware processes fast.	In-house engineering cost of development, royalty payments for an IP core of processor and time-to-market cost.
Dual core processor	To significantly enhance the performance of the system.	Reduced engineering cost.	Manufacturing cost, as dual core processors are costly.
Accelerator	To accelerate the execution of codes. A floating point coprocessor accelerates mathematical operations and Java accelerator accelerates Java code execution.	Increases performance by co-processing with the main processor.	Engineering cost of development or royalty payments for IP core of processor and time-to-market cost.

A DSP for mobile phones, for example, OMAP of Texas Instruments, uses the effective power dissipation methods of dynamic switching both for power supply voltage and operating frequency of the CPU core.

For a number of applications, the DSPs cores may not suffice. Domain specific ASIPs have specific instruction sets. For IOs, network, media or security applications, smart card, video game, palm top computer, cell phone, mobile-Internet, hand-held embedded systems, Gbps transceivers, Gbps LAN systems, satellite or missile systems, we need special processing units in a VLSI circuit designed to function as a processor with an instruction-set for programmability. These special units are called domain-specific ASIP.

1.7.8 Embedding ARM processor

Examples of Stream 4 GPPs in Table 1.4 are ARM 7 and ARM 9. The core of these processors can be embedded onto a VLSI chip or an SoC. An ARM-processor VLSI-architecture is available either as a CPU chip or for integrating it into VLSI or SoC. ARM, Intel and Texas Instruments and several other companies have developed such processors. ARM provides CISC functionality with RISC architecture at the core. 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].

ARM integrates with other features (for example DSP) in new GPPs, which are available from several sources, for example, Intel and Texas Instruments. Exemplary ARM 9 applications are setup boxes, cable modems, and wireless-devices such as mobile handsets.

ARM9 has a single cycle 16×32 multiple accumulate unit. It operates at 200 MHz. It uses 0.15 μm GS30 CMOSs. It has a five-stage pipeline. It incorporates RISC core with CISC functions. It integrates with a DSP when designed for an ASIC solution. An example is its integration with DSP is TMS320C55x from Texas Instruments. [Refer to http:/www.ti.com/sc/docs/asic/modules/arm7.htm and arm9.htm]

A lower performance but very popular version of ARM9 is ARM7. It operates at 80 MHz. It uses 0.18 μm based GS20 μm CMOSs. Using ARM7, ARM9 and CORTEX-M3, a large number of embedded systems have recently become available.

Lately, a new class of embedded systems has emerged that additionally incorporates ASSP chips or cores in its design.

1.7.9 Embedding ASSP

Assume that there is an embedded system for real-time video processing. Real-time processing arises for digital television, high definition TV decoders, set-up boxes, DVD (Digital Video Disc) players, web phones, video-conferencing and other systems. An ASSP that is dedicated to these specific tasks alone provides a faster solution. The ASSP is configured and interfaced with the rest of the embedded system.

Assume that there is an embedded system that using a specific protocol interconnects, its units through specific bus architecture to another system. Also, assume that suitable encryption and decryption is required. [The output bit stream encryption protects messages or design from passing to an unknown external entity.] For these tasks, besides embedding the software, it may also be necessary to embed some RTOS features [Section 1.4.6]. If the software alone is used for the above tasks, it may take a longer time than a hardwired solution for application-specific processing. An ASSP chip provides such a solution. For example, an ASSP chip [from i2Chip (http://www.i2Chip.com)] has a TCP, UDP, IP, ARP and Ethernet 10/100 MAC (Media Access Control) hardwired logic included into it. The chip from i2Chip, W3100A, is a unique hardwired Internet connectivity solution. Much needed TCP/IP stack processing software for networking tasks is thus available as a hardwired solution. This gives output five times faster than a software solution using the system's GPP. It is also an RTOS-less solution. Using the same microcontroller in the embedded system to which this ASSP chip

interfaces, Ethernet connectivity can be added. Another ASSP, which is now available, is the 'Serial-to-Ethernet Converter (IIM7100). It does real-time data processing by a hardware protocol stack. It needs no change in the application software or firmware and provides the most economical and smallest RTOS-solution.

An ASSP is used as an additional processing unit for running application specific tasks in place of processing using embedded software.

1.8 DESIGN PROCESS IN EMBEDDED SYSTEM

The concepts used during a design process are as follows.

1. *Abstraction:* Each problem component is first abstracted. For example, in the design of a robotic system, the problem of abstraction can be in terms of control of arms and motors.
2. *Hardware and Software architecture:* Architectures should be well understood before a design.
3. *Extra functional Properties:* Extra functionalities required in the system being developed should be well understood from the design.
4. *System Related Family of designs:* Families of related systems developed earlier should be taken into consideration during designing.
5. *Modular Design:* Modular design concepts should be used. System designing is fast by decomposition of software into modules that are to be implemented. Modules should be such that they can be composed (coupled or integrated) later. Effective modular design should ensure effective (i) function independence, (ii) cohesion and (iii) coupling.
   (a) Modules should be clearly understood and should maintain continuity.
   (b) Also, appropriate protection strategies are necessary for each module. A module is not permitted to change or modify another module functionality. For example, protection from a device driver modifying the configuration of another device.
6. *Mapping:* Mapping into various representations is done from software requirements. For example, data flow in the same path during the program flow can be mapped together as a single entity. Transform and transaction mapping design processes are used in designing. For example, an image is input data to a system; it can have a different number of pixels and colours. The system does not process each pixel and colour individually. Transform mapping of image is done by appropriate compression and storage algorithms. Transaction mapping is done to define the sequence of images.
7. *User Interface Design:* User interface design is an important part of design. User interfaces are designed as per user requirements, analysis of the environment and system functions. For example, in an automatic chocolate vending machine (ACVM) system, the user interface is an LCD multiline graphics display. It can display a welcome message as well as specify the coins needed to be inserted into the machine for each type of chocolate. The same ACVM may be designed with touchscreen User Interface (GUI), or it may be designed with Voice User Interfaces (VUIs). Any of these interface designs has to be validated by the customer. For example, the ACVM customer who installs the machine must validate message language and messages to be displayed before an interface design can proceed to the implementation stage.
8. *Refinements:* Each component and module design needs to be refined iteratively till it becomes the most appropriate for implementation by the software team.

The software design process may require use of Architecture Description Language (ADL). It is used for representing the following: (i) Control Hierarchy (ii) Structural Partitioning (iii) Data Structure and Hierarchy (iv) Software Procedures.

Figure 1.11 shows the activities for software-design cycle during an embedded software-development process and the cycle may be repeated till tests show the verification of specifications.

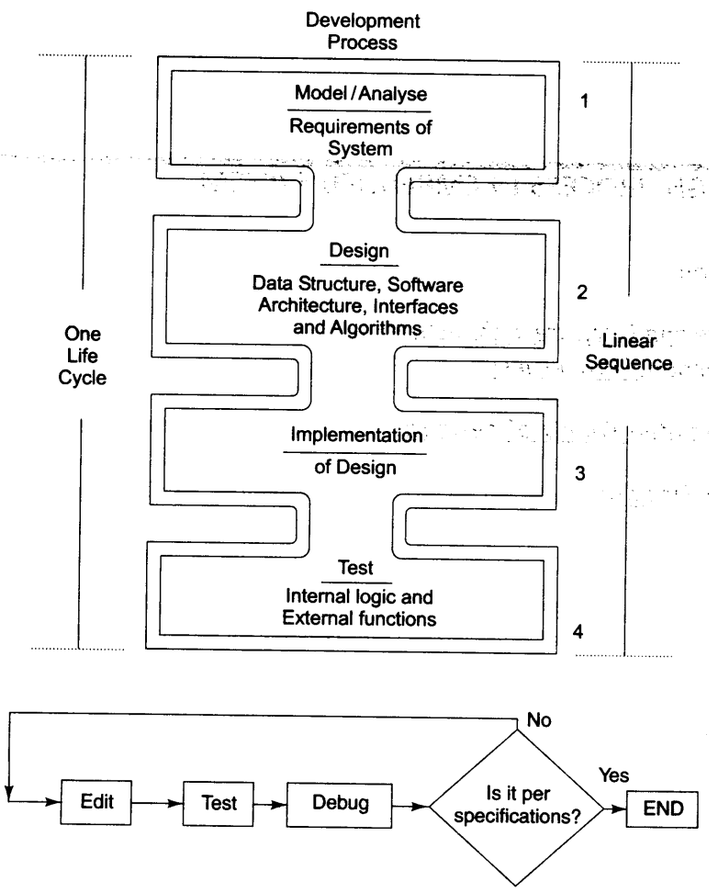


Fig. 1.11 Activities for software design during an embedded software-development process

1.8.1 Design Metrics

A design process takes into account design metrics. There are several design metrics for an embedded system, and these are listed in Table 1.8.

1.8.2 Abstraction of Steps in the Design Process

A design process is called bottom-to-top design if it builds by starting from the components. A design process is called top-to-down design if it first starts with abstraction of the process and then after abstraction the details are created. Top-to-down design approach is the most favoured approach. The following lists the five levels of abstraction from top to bottom in the design process:

(1) ***Requirements:*** Definition and analysis of system requirement. It is only by a complete clarity of the required *purpose, inputs, outputs, functioning, design metrics* (Table 1.8) and *validation requirements* for finally developed systems specifications that a well designed system can be created. There has to be consistency in the requirements.

Table 1.8 Design metrics used in the embedded systems

Design Metrics	*Description*
Power Dissipation	For many systems, particularly battery operated systems, such as mobile phone or digital camera the power consumed by the system is an important feature. The battery needs to be recharged less frequently if power dissipation is small.
Performance	Instructions execution time in the system measures the performance. Smaller execution time means higher performance. For example, a mobile phone, voice signals processed between antenna and speaker in 0.1s shows phone performance. Consider another. For example, a digital camera, shooting a 4M pixel still image in 0.5s shows the camera performance.
Process deadlines	There are number of processes in the system, for example, keypad input processing, graphic display refresh, audio signals processing and video signals processing. These have deadlines within which each of them may be required to finish computations and give results.
User interfaces	These include keypad GUIs and VUIs.
Size	Size of the system is measured in terms of (i) physical space required, (ii) RAM in kB and internal flash memory requirements in MB or GB for running the software and for data storage and (iii) number of million logic gates in the hardware.
Engineering cost	Initial cost of developing, debugging and testing the hardware and software is called engineering cost and is a one-time non-recurring cost.
Manufacturing cost	Cost of manufacturing each unit.
Flexibility	Flexibility in design enables, without any significant engineering cost, development of different versions of a product and advanced versions later on. For example, software enhancement by adding extra functions necessitated by changing environment and software re-engineering.
Prototype development time	Time taken in days or months for developing the prototype and in-house testing for system functionalities. It includes engineering time and making the prototype time.
Time-to-market	Time taken in days or months after prototype development to put a product for users and consumers.
System and user safety	System safety in terms of accidental fall from hand or table, theft (e.g., a phone locking ability and tracing ability) and in terms of user safety when using a product (for example, automobile brake or engine).
Maintenance	Maintenance means changeability and additions to the system; for example, adding or updating software, data and hardware. Example of software maintenance is additional service or functionality software. Example of data maintenance is additional ring-tones, wallpapers, video-clips in mobile phone or extending card expiry date in case of smart card. Example of hardware maintenance is additional memory or changing the memory stick in mobile computer and digital camera.

(2) ***Specifications:*** Clear specifications of the required system are must. Specifications need to be precise. Specifications guide customer expectations from the product. They also guide system architecture. The designer needs specifications for (i) hardware, for example, peripherals, devices processor and memory specifications, (ii) data types and processing specifications, (iii) expected system behaviour specifications, (iv) constraints of design, and (v) expected life cycle specifications. Process specifications are analysed by making lists of inputs on events, outputs on events and how the processes activate on each event (interrupt).

(3) ***Architecture:*** Data modeling designs of attributes of data structure, data flow graphs (Section 6.2), program models (Section 6.1), software architecture layers and hardware architecture are defined. Software architectural layers are as follows:

1. The first layer is an architectural design. Here, a design for system architecture is developed. The question arises as to how the different elements—data structures, databases, algorithms, control functions, state transition functions, process, data and program flow—are to be organised.
2. The second layer consists of data-design. Questions at this stage are as follows. What design of data structures and databases would be most appropriate for the given problem? Whether data organised as a tree- like structure will be appropriate? What will be the design of the components in the data? [For example, video information will have two components, image and sound.]
3. The third layer consists of interface design. Important questions at this stage are as follows. What shall be the interfaces to integrate the components? What is the design for system integration? What shall be design of interfaces used for taking inputs from the data objects, structures and databases and for delivering outputs? What will be the port structure for receiving inputs and transmitting outputs?

(4) ***Components:*** The fourth layer is a component level design. The question at this stage is as follows. What shall be the design of each component? There is an additional requirement in the design of embedded systems, that each component should be optimised for memory usage and power dissipation. Components of hardware, processes, interfaces and algorithms. The following lists the common hardware components:

1. Processor, ASIP and single purpose processors in the system
2. Memory RAM, ROM or internal and external flash or secondary memory in the system
3. Peripherals and devices internal and external to the system
4. Ports and buses in the system
5. Power source or battery in the system

During software development process we can model the components as object-oriented. Table 1.9 lists the stages as components-based object-oriented software development process.

(5) ***System Integration:*** Built components are integrated in the system. Components may work fine independently, but when integrated may not fullfil the design metrics. The system is made to function and validated. Appropriate tests are chosen. Debugging tools are used to correct erroneous functioning.

Each component and its interface system is integrated after the design stage. Program implementation is in a language and may use an integrated development environment (IDE), and source code engineering tools, which should follow the model, software architecture and design specifications. Program simplicity should be maintained during the implementation process.

The design stages range from abstraction to detailed designing to verification activities. Continuous refinement in design can be made by effective communication between designers and implementers. Software design can be assumed to consist of four layers: architecture design, data design, interfaces design and component level design.

Table 1.9 Components-based object-oriented software development process

Effort	*Activities*	*Model Deficiency*
Stage 1	Components that could be used in software development identified	Need for robust interfaces and slow development in case the reusable components are not available in required numbers
Stage 2	Selection of available classes (single logically bonded groups) from a software components resource library	
Stage 3	Sort components, which are available and reusable by re-engineering and which are unavailable	
Stage 4	Re-engineer components and create unavailable components	
Stage 5	Construct software from the components and test them	
Stage 6	Iteratively construct till final validation of software	

Actions at each step Research by software engineering experts have shown that on an average, a designer needs to spend about 50% of the time for planning, analysis and design, 40% for testing, validation and debugging and 10–15% on coding. Action required to be taken at each step in the design process is listed in Table 1.10.

Table 1.10 Action to be taken at each step of design process

Design Metrics	*Description*
Analysis	Design is analyzed
Steps for improvement	The result of analysis is used to improve design to meet specifications and metrics
Verification	System design must be verified to ensure that it meets the design metrics given in Table 1.8

1.8.3 Challenges in Embedded System Design: Optimizing Design Metrics

Following are the challenges that arise during the design process.

Amount and type of hardware needed: Optimizing the requirement of microprocessors, ASIPs and single purpose processors in the system on the basis of performance, power dissipation, cost and other design metrics are the challenges in a system design. A designer also chooses the appropriate hardware (memory RAM, ROM or internal and external flash or secondary memory, peripherals and devices internal and external ports and buses and power source or battery) taking into account the design metrics given in Table 1.8; for example, power dissipation, physical size, number of gates and the engineering, prototype development and manufacturing costs.

Optimizing Power Dissipation and Consumption: Power, consumption during the operational and idle state of system should be optimal. The following methods are used to meet the design challenges.

Clock Rate Reduction Power dissipation typically reduces 2.5 μW per 100 kHz of reduced clock rate. So reduction from 8000 kHz to 100 kHz reduces power dissipation by about 200 μW, which is nearly similar to when the clock is nonfunctional. [Remember, total power dissipated (energy required) may not reduce. This is because on reducing the clock rate, the computations will take a longer time and total energy required equals the power dissipation per second multiplied by computation time].

The power 25 μW is typically the residual dissipation needed to operate the timers and few other units. By operating the clock at a lower frequency or during the power-down mode of the processor, the advantages are as follows: (i) Power loss due to heat generation reduces. (ii) Radio frequency interference also reduces due to the reduced power dissipation within the gates. [Radiated RF (Radio Frequency) power depends on the RF current inside a gate, which reduces due to increase in 'ON' state resistance between drain and channel of each MOSFET transistor and that reduces heat generation.]

Voltage Reduction In portable or hand-held devices such as a cellular phone, compared to 5 V operation, a CMOS circuit power dissipation reduces by one sixth, ~$(2V/5V)^2$, in 2.0 V operation. Thus the time intervals needed for recharging the battery increase by a factor of six.

Wait, Stop and Cache Disable Instructions An embedded system may need to be run continuously, without being switched off; the system design, therefore, is constrained by the need to limit power dissipation while it is ON but is in idle state. Total power consumption by the system while in running, waiting and idle states should be limited. A microcontroller must provide for executing *Wait* and *Stop* instructions for the power-down mode. One way to reduce power dissipation is to cleverly incorporate into software the *Wait* and *Stop* instructions. Another is to operate the system at the lowest voltage levels in the idle state and selecting power-down mode in that state. Yet another method is to disable use of certain structural units of the processor—for example, caches—when not necessary and to keep in disconnected state those structure units that are not needed during a particular software execution, for example timers or IO units.

Operations can be performed at low voltage or reduced clock rate in order to control power dissipation. For embedded system software, performance analysis during its design phase must also include the analysis of power dissipation during program execution and during standby. An embedded system has to perform tasks continuously from power-up to power-off and may even be kept 'ON' continuously. Clever real-time programming by using 'Wait' and 'Stop' instructions and disabling certain units when not needed is one method of saving power during program execution.

Process Deadlines Meeting the deadline of all processes in the system while keeping the memory, power dissipation, processor clock rate and cost at minimum is a challenge.

Flexibility and Upgrade ability Flexibility and upgrade ability in design while keeping the cost minimum and without any significant engineering cost is a challenge. Flexibility and upgrade ability allow different and advanced versions of a product to be introduced in the market later on.

Reliability Designing a reliable product by appropriate design, testing and thorough verification, is a challenge. The goal of testing is to find errors and to validate that the implemented software is as per the specifications and requirements. Verification refers to an activity to ensure that specific functions are correctly implemented. Validation refers to an activity to ensure that the system that has been created is as per the requirements agreed upon at the analysis phase, and to ensure its quality.

1.9 FORMALIZATION OF SYSTEM DESIGN

Formalization of system design is done using a top-down approach by abstraction (Section 1.8.2) and by

- Detailing requirements and specifications of hardware and software

- Defining architectures of hardware and software
- Coding and implementation as per architecture
- Testing, validation and verification of system

Since a diagrammatic model clears the design concepts better than abstraction, a modeling language, for formalization can be used. The Universal Modeling Language (UML) is used. In UML, a designer describes the following:

1. 'User Diagram', 'Object Diagram', 'Sequence Diagram', 'State Diagram', 'Class Diagram' and 'Activity Diagram'
2. Classes and Objects, which describe *identity*, *attributes*, *components* and *behaviour*
3. Inheritances of the classes and objects
4. Interfaces of the objects and their implementation at the objects
5. Structural description of the design components
6. Behavioral description in terms of states, state machine and signals (Section 6.3)
7. Events description

Section 6.5 will describe UML in detail. Chapters 11 and 12 will describe the model design examples in detail.

1.10 DESIGN PROCESS AND DESIGN EXAMPLES

1.10.1 System Design Process Examples

Chapters 11 and 12 will describe design examples in detail.

1.10.2 Automatic Chocolate Vending Machine (ACVM)

Let us consider an automatic chocolate vending machine. This interesting example given here helps a reader to understand several concepts of programming an embedded system as a multitasking system.

Figure 1.12 shows the diagrammatic representation of ACVM. Assume that ACVM has following components:

1. It has 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.
2. It has an LCD display unit on the top of the machine. It displays menus, text entered into the ACVM and pictograms, welcome, thank you and other messages. It enables the child as well as the ACVM owner to graphically interact with the machine. It also displays time and date. (For GUIs, the keypad and LCD display units or touch screen are basic units.)
3. It has a coin insertion slot and a mechanical coin sorter so that child can insert coins to buy a chocolate.
4. It has a delivery slot so that child can collect the chocolate and coins, if refunded.
5. It has an Internet connection port using a USB based wireless modem so that owner can know status of the ACVM sales from a remote location.

ACVM Functions Assume that ACVM functions are as follows:

1. The ACVM displays the GUIs and if the child wishes to enter contact information, birthday information or get answer to FAQs, it displays the appropriate menu.
2. It displays a welcome message when in idle state. It also continuously displays time and date at the right bottom corner of display screen. It can also intermittently display news, weather data or advertisements or important information of interest during idle state.

3. When first coin is inserted, a timer also starts. The child is expected to insert all required coins in 2 minutes.
4. After 2 minutes the ACVM will display a query to the child if the child does not insert sufficient coins. If the query is not answered the coins are refunded.
5. Within 2 minutes if sufficient coins are collected, it displays the message, 'Thanks, wait for few moments please!', delivers the chocolate through the delivery slot and displays message, 'Collect the chocolate and visit again, please!'

Hardware units ACVM embeds the following hardware units.

1. Microcontroller or ASIP (Application Specific Instruction Set Processor)
2. RAM for storing temporary variables and stack
3. ROM for application codes and RTOS codes for scheduling the tasks
4. Flash memory for storing user preferences, contact data, user address, user date of birth, user identification code, answers of FAQs
5. Timer and interrupt controller
6. A TCP/IP port (Internet broadband connection) to the ACVM for remote control and for the owner to get ACVM status reports
7. ACVM specific hardware to sort coins of different denominations. Each denomination coin generates a set of status and input bits and port-interrupts. Using an ISR for that port, the ACVM processor reads the port status and input bits. The bits give the information about which coin has been inserted. After each read operation, the status bits are reset by the routine
8. Power supply

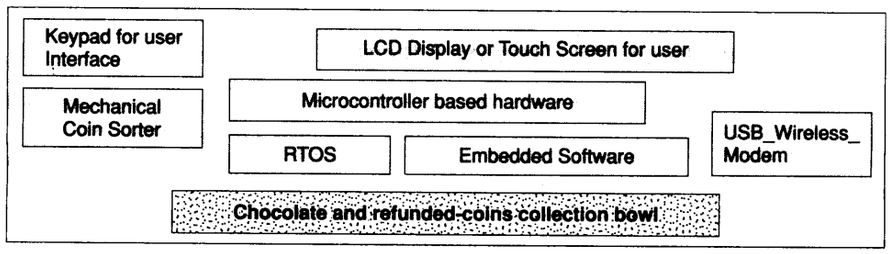


Fig. 1.12 Diagrammatic representation of the ACVM

Software components ACVM embeds the following software components:

1. Keypad input read task
2. Display task
3. Read coins task for finding coins sorted
4. Deliver chocolate task
5. TCP/IP stack processing task
6. TCP/IP stack communication task

1.10.3 Smart Card

Smart card is one of the most used embedded system today. It is used for credit–debit bankcard, ATM card, e-purse or e-Wallet card, identification card, medical card (for history and diagnosis details) and card for a

number of new innovative applications. [Reader may refer to a frequently updated website, http://www.sguthery@tiac.net for the answers of frequently asked questions about cards.] The security aspect is of paramount importance for smart card use, when used for financial and banking-related transactions. [Reader may refer to http://www.home.hkstar.com/~alanchan/papers/smartCardSecurity/ and http://www.research.ibm.com/secure_systems/scard.htm for details of the card-security requirements.]

The smart card is a plastic card ISO standard dimensions, 85.60 × 53.98 × 0.80 mm. It is an embedded system on a card: SoC (System-On-Chip). ISO recommended standards are ISO7816 (1 to 4) for host-machine contact-based cards and ISO14443 (Part A or B) for the contactless cards. The silicon chip is just a few multimeters in size and is concealed in-between the layers. Its very small size protects the card from bending. Figure 1.13 shows embedded-system hardware components for a contactless smart card.

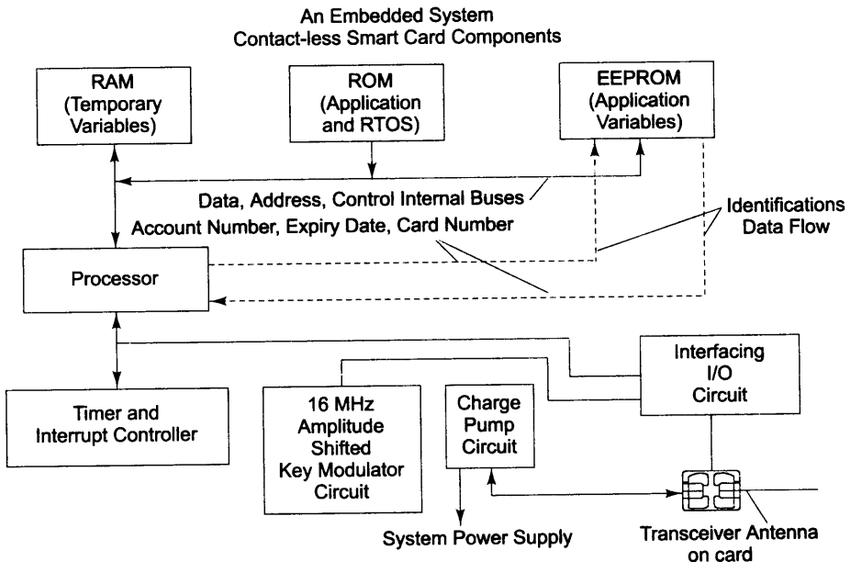


Fig. 1.13 Embedded hardware components in a contact less smart card

Embedded Hardware The embedded hardware components are as follows:

- Microcontroller or ASIP
- RAM for temporary variables and stack
- One time programmable ROM for application codes and RTOS codes for scheduling the tasks
- Flash for storing user data, user address, user identification codes, card number and expiry date
- Timer and interrupt controller
- A carrier frequency ~16 MHz generating circuit and Amplitude Shifted Key (ASK) modulator
- Interfacing circuit for the IOs
- Charge pump for delivering power to the antenna for transmission and for system circuits. The charge pump stores charge from received RF (radio frequency) at the card antenna in its vicinity. [The charge pump is a simple circuit that consists of the diode and high value ferroelectrics material-based capacitor.]

The details of the basic hardware units are as follows:

1. The **Microcontroller** used can be MC68HC11D0 or PIC16C84 or a smart card processor Philips Smart XA or a similar ASIP Processor. MC68HC11D0 has 8 kB internal RAM and 32 kB EPROM and 2/3 wire protected memory. Most cards use 8-bit CPUs. The recent introduction in the cards is of a 32-bit RISC CPU. A smart card CPU should have special features, for example, a security lock. The lock is for a certain sections of the memory. A protection bit at the microcontroller may protect 1 kB or more data from modification and access by any external source or instructions outside that memory. Once the protection bit is placed at the maskable ROM in the microcontroller, the instructions or data within that part of the memory are accessible from instructions in that part only (internally) and not accessible from the external instructions or instructions outside that part. The CPU may disable access by blocking the write cycle placement of the data bits on the buses for instructions and data protection at the physical memory after certain phases of card initialization and before issuing the card to the user. Another way of protecting is as follows: The CPU may access by using the physical addresses, which are different from the logical address used in the program.
2. **ROM** is used in the card. The usual size is 8 or 64 kB for usual or advanced cryptographic features in the card, respectively. Full or part of ROM bus activates only after a security check. The processor protects a part of the memory from access. The ROM stores the following.
   i. Fabrication key, which is a unique secret key for each card. It is inserted during fabrication.
   ii. Personalization key, which is inserted after the chip is tested on a printed circuit board. Physical addresses are used in the testing phase. The key preserves the fabrication key and this key insertion preserves the card personalization. After insertion of this key, RTOS and applications use only logical addresses.
   iii. RTOS codes
   iv. Application codes
   v. A utilization lock to prevent modification of two PINs and to prevent access to the OS and application instructions. It stores after the card enters the utilization phase.
3. **EEPROM or Flash** is scalable. These means that only that part of the memory required for a particular operation will unlock for use. The authorizer will use the required part; the application will use the other part. It is protected by the access conditions stored therein. It stores the following:
   i. PIN (Personal Identification Number), the allotment and writing of which is by the authorizer (for example, a bank) and its use is possible by the latter only by using the personalization and fabrication keys. It is for identifying the card user in future transactions. Card user is given this key. Alternatively, a modifiable password is given to the user and password opens the PIN key.
   ii. An unblocking PIN for use by the authorizer (say the bank). Through this key, the card circuit identifies the authorizer before unblocking. Data of the user unblocks for the authorizer and storing of information on the card is possible by the authorizer through the host.
   iii. Access conditions for various hierarchically arranged data files.
   iv. Card user data, for example, name, bank and branch identification number and account number or health insurance details.
   v. Data post issue that the *application* generates. For example, in case of e-purse, the details of previous transactions and current balance. Medical history and diagnosis details and/or previous insurance claims and pending insurance claims record in case of a medical card.
   vi. It also stores the application's non-volatile data.
   vii. Invalidation lock sent by the host after the expiry period or card misuse and user account closing request. It locks the data files of the master or elementary individual file or both.
4. **RAM** stores the temporary variables and stack during card operations by running the OS and the *application*.

5. **Chip power supply** voltage extracts by a charge pump circuit. The pump extracts the charge from the signals from the host analogous to what a mouse does in a computer and delivers the regulated voltage to the card chip, memory and IO system. Signals can be from antenna or from clock pin. In a typical card operation using 0.18 μm technology, 1.6 to 5.5 V is the threshold limit and for a 0.35 μm technology, 2.7 to 5.5 V.
6. **IO System** of chip and host interact through asynchronous serial UART (Section 3.2.3) at 9.6 k or 106 k or 115.2 k baud/s. The chip interconnects to a card hosting system (reader and writer) either through the gold contacts or through a centimeter sized antenna on each side. The latter provides *contactless* interconnection between the IO pins, which are meant for *contact-based* interaction, RST (Reset Signal from host) and Clock (from host).
7. **Wireless Communication** for IO interaction is by radiations through the antenna coils for contactless interaction. The card and host interact through a card modem and a host modem. The application protocol data unit (APDU) is a standard for communication between the card and host computer. Modulation is with 10% index amplitude modulating carrier of 13.66–13.56 Mbps ASK (amplitude shifted keying) is used for contactless communication at data rates of ~1 Mbps. One-sixteenth frequency subcarrier modulates through BPSK (Binary Phase Shifted Keying).

Embedded Software Smart card embeds the following software components:

1. Boot-up, initialisation and OS programs
2. Smart card secure file system
3. Connection establishment and termination
4. Communication with host
5. Cryptography algorithm
6. Host authentication
7. Card authentication
8. Saving addition parameters or recent new data sent by the host (for example, present balance left)

The smart card is an exemplary secure embedded system with security software. The card needs cryptographic software. Embedded software in the card needs special features in its operating system over and above the MS DOS or UNIX system features. Special features needed are as follows:

1. *Protected environment.* It means software should be stored in the protected part of the ROM.
2. *Restricted run-time environment.*
3. Its *OS*, every method, class and run time library *should be scalable.*
4. Code-size *generated should be optimum.* The system needs should not exceed 64 kB memory.
5. Limited use of data types; multidimensional arrays, long 64-bit integer and floating points and very limited use of the error handlers, exceptions (Section 4.2.2), signals (Sections 6.5 and 7.10), serialization, debugging and profiling. [Serialization is the process of converting an object into a data stream for transferring it to network or from one process to another. The de-serialized data are the receiver end].
6. A three-layered file system for the data. One file for the *master file* to store all file headers. A header means file status, access conditions and the file lock. The second file is a *dedicated file* to hold a file grouping and headers of the immediate successor elementary files of the group. The third file is the *elementary file* to hold the file header and its file data.
7. There is either a fixed length file management or a variable length file management with each file having a predefined offset.
8. Classes for the network, sockets, connections, data grams, character-input output and streams, security management, digital-certification, symmetric and asymmetric keys-based cryptography and digital signatures.

1.10.4 Digital Camera

Digital cameras may have 4 to 6 M pixel still images, clear visual display (ClearVid) CMOS sensor, 7 cm wide LCD photo display screen, enhanced imaging processor, double anti blur solution and high-speed processing engine, 10X optical and 20X digital zooms and can also record high definition video-clips. It therefore has speaker microphone(s) for high quality recorded sound. It has an audio/video out port for connecting to a TV/DVD player or computer.

Let us assume that the camera is still just a camera. Figures 1.14(a) and (b) show hardware and software components in a simple digital camera. Assume that the camera has the following components:

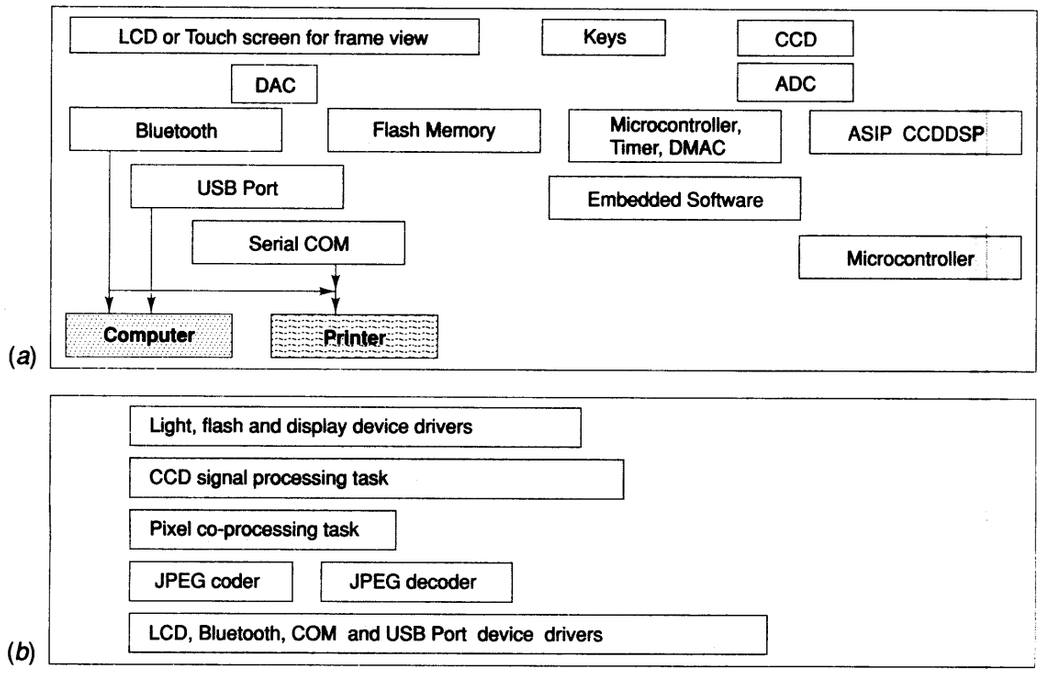


Fig. 1.14 (*a*) Digital camera hardware components (*b*) Digital camera software components

1. It has keys on the camera. That enables a user to operate the camera. It has navigation keys to navigate through the images back and forth.
2. Shutter, lens and charge coupled device (CCD) array sensors for images in sizes 2592 × 1944 pixels = 5038848 pixels, VGA (E-mail) 640 × 480 = 307200 pixels, 2592 × 1728 = 3.2 M pixels, 2048 × 1536 pixels = 3 M pixels, or 1280 × 960 pixels = 1 M pixels.
3. It has a good resolution photo quality LCD display unit on the back of camera to show photographs or recorded video-clips. It displays text such as image-title, shooting data and time and serial number. It displays messages. It displays the GUI menus when the user interacts with the camera.
4. It has a self-timer lamp for flash.
5. Internal memory flash to store OS and embedded software, and limited number of image files.
6. Flash memory stick of 2 GB or more for large storage.

7. It has Universal Serial Bus (USB) port (Section 3.10.3) or Bluetooth interface, which connects it to a computer and printer.

Camera Functions Assume that the camera functions is as follows:

1. It displays the frame view on the LCD screen so that user can adjust the camera inclination before shooting the frame.
2. It displays the saved images on the LCD using navigation keys.
3. When a key for opening the shutter is pressed, the flash lamp glows and the self-timer circuit switches off the lamp automatically.
4. The frame light falls on the CCD array, which transmits the bits for each pixel in each row in the frame through an ADC. Bits from dark area pixels in each row are used for offset corrections in the CCD signal for light intensities in each row.
5. The CCD bits of each pixel in each row and column are offset corrected using a CCD signal processor (CCDSP).
6. The processed signals are compressed using a JPEG CODEC and saved in one jpg file for each frame. A DSP does compression using the the discrete cosine transformations (DCTs) and decompression by inverse DCT. Thereafter, it also does Huffman coding for JPEG compression.
7. A DAC sends the inputs for the display unit. The DAC gets the input from the pixel processor, which gets the inputs from the JPEG files for the saved images and gets input directly from the CCDSP through the pixel processor or the frame in the present view.

Digital Hardware units The camera embeds the following hardware units.

1. Microcontroller or ASIP
2. Multiple processors (CCDSP, DSP, pixel processor and others)
3. RAM for storing temporary variables and stack
4. ROM for application codes and RTOS codes for scheduling tasks
5. Timer, flash memory for storing user preferences, contact data, user address, user date of birth, user identification code, ADC, DAC and interrupt controller (Sections 1.3.3, 1.3.5, 1.3.7 and 1.3.11)
6. USB controller (Section 3.10.3)
7. Direct memory access controller (Section 4.8)
8. LCD controller (Section 3.3.4)
9. Battery

Software components The camera embeds the following software components:

1. CCD signal processing for off-set correction
2. JPEG coding
3. JPEG decoding
4. Pixel processing before display
5. Memory and file systems
6. Light, flash and display device drivers
7. COM, USB port and Bluetooth device drivers for port operations for printer and computer communication control

1.10.5 Mobile Phone

The mobile phone today has a large number of features. It has sophisticated hardware and software.

Hardware units A mobile phone embeds an SoC (System-on-Chip) integrating the following hardware units.

1. Microcontroller or ASIP [An ASIP is configured to process encoding and deciphering and another does the voice compression. Third ASIC dials, modulates, demodulates, interfaces the keyboard and touch screen or multiple line LCD graphic displays, and processes the data input and recall of data from memory].
2. DSP core, CCDSP, DSP, video, voice and pixel processors
3. Flash, memory stick, EEPROMs and SRAMs
4. Peripheral circuits, ADC, DAC and interrupt controller
5. Direct memory access controller (Section 4.8)
6. LCD controller (Section 3.3.4)
7. Battery

Software components The mobile phone software development tools are as follows:

1. OS (Windows Mobile, Palm, Symbian) or BREW
2. Java 2 Micro Edition (J2ME) along with KVM as a Java Virtual Machine (Section 5.7.4)
3. Java Wireless toolkit with JDK (Java Development Kit)

The mobile phone embeds the following software components:

1. Memory and file systems
2. Keypad, LCD, serial, USB, 3G or 2G port device drivers for port operations for keypad, printer and computer communication control
3. SMS (Short Messaging Service) message creation and communicator, contact and PIM (personal information manager), task-to-do manager and email
4. Mobile imager for uploading pictures and MMS (multimedia messaging service)
5. Mobile browser for access to the Web
6. Downloader for Java games, ring-tones, games, wall papers
7. Simple camera with (Section 1.10.4)
8. Bluetooth synchronization, IrDA and WAP connections support (Section 3.13)

1.10.6 Mobile Computer

The mobile computer has Windows CE or Windows mobile as OS. It has a touch screen for GUI. The user uses a stylus to enter commands. It has a virtual keypad (the keypad displayed on the screen and entries of text and commands is through the stylus.

In addition to phone, a mobile computer has following software components:

1. OS (Windows CE, Windows Mobile, PocketPC, Palm OS or Symbian OS)
2. Touch screen GUIs, memory and file systems
3. Memory stick
4. Outlook, Internet explorer, Word, Excel, Powerpoint, and handwritten text processor
5. Applications or enterprise software

1.10.7 A Set of Robots

Consider a set of 8 robots. One robot is the master robot (music director) and seven are slave robots (conductors). Assume that the set is used to play an orchestra. Figures 1.14(a) and (b) show hardware and software components in the set of robots. Assume that the robot has the following components.

1. The master robot signals the commands and slave robots play accordingly.

2. Each robot is assumed to have five degrees of freedom. Each robot has a mechanical system of five degrees of freedom. At each degree of freedom, there is a servomotor. A servomotor controls by PWM method (Section 1.3.7). Each motor is controlled in a sequence to let the robot perform the desired action.
3. Each robot has a microcontroller with expansion ports, P0, ..., P8. Actually a single ASIC can perform multiple port functions of a microcontroller. However since the engineering cost of ASIC development is high, a general purpose microcontroller 68HC12 or 8051 is used.
4. The port outputs connect the motors and PWM outputs drive the motors in each robot.
5. Each robot has a serial IO with IrDA protocol. (Section 3.13.1)
6. Internal memory flash to store the OS, embedded software and limited amount of music.
7. There is a music file processor for playing the music. Slave robots have speaker outputs for playing music.

Master Robot Functions Assume that master robot functioning is as follows:

1. It receives commands from a remote controller to start and stop the music and the code for the specific orchestra to be played.
2. It sends PWM signals to the ports for moving the sticks in both hands as per the program.
3. It establishes and binds the sockets (the virtual devices) connection with the slaves. It sends the signals through sockets using IrDA protocols. The byte streams response to the clients are as per the music file to be played by the slave.

Slave Robot Functions Assume that slave robot functioning is as follows:

1. It establishes and binds the sockets (the virtual devices) connection with the master.
2. It receives from a master socket the commands accept ( ) and write ( ); it also receives commands to start and stop music and the code for the specific orchestra to be played.
3. It receives the signals through sockets using IrDA protocols. The byte streams from the server are as per the music file being played.

Hardware units Robots embed the following hardware units.

1. Microcontroller or ASIP
2. Music file processor
3. RAM for storing temporary variables and stack
4. ROM for application codes and RTOS codes for scheduling robot actions and tasks
5. Timer, flash memory for storing user preferences and music files
6. IrDA controller (Section 3.13.1)
7. Direct memory access controller (Section 4.8)
8. Power supply source or battery

Software components Robots embed the following software components:

1. Socket functions
2. Music coding
3. Music decoding
4. Memory and file systems
5. Light, flash and display device drivers
6. IrDA and socket port device drivers
7. Motor drivers
8. IO ISRs

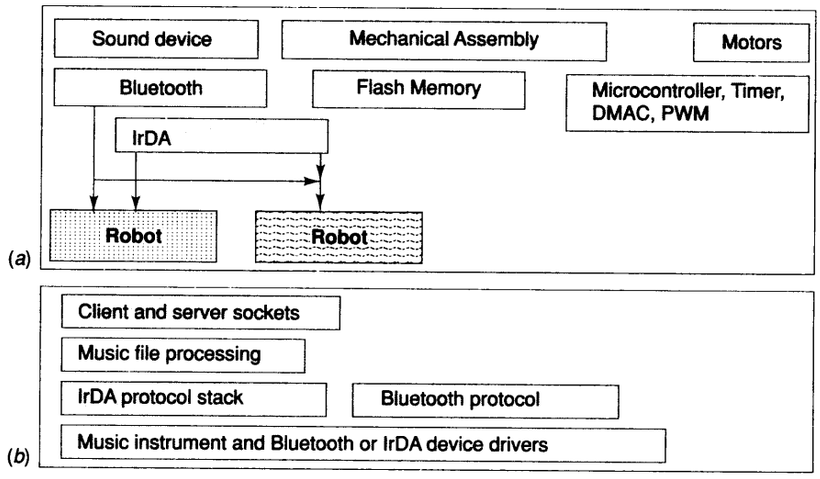


Fig. 1.15 (*a*) Hardware components in the set of robots (*b*) software components in the set of robots in which a master robots signals the commands and slave robots play according to the signals from the master

1.11 CLASSIFICATION OF EMBEDDED SYSTEMS

We can classify embedded systems into three types as follows.

1. ***Small scale embedded systems:*** These systems are designed with a single 8- or 16-bit microcontroller; they have little hardware and software complexities and involve board-level design. They may even be battery operated. When developing embedded software for these, an editor, assembler and cross assembler, an integrated development environment (ISE) tool specific to the microcontroller or processor used, are the main programming tools. Using 'C' language, programs are compiled into the assembly and executable codes are appropriately located in the system memory. The software has to fit within the memory available and keep in view the need to limit power dissipation when the system is running continuously.
2. ***Medium scale embedded systems:*** These systems are usually designed with a single or a few 16- or 32-bit microcontrollers, DSPs or RISCs. These systems may also employ the readily available single purpose processors and IPs (explained later) for the various functions—for example, bus interfacing. [ASSPs and IPs may also have to be appropriately configured by the system software before being integrated into the system-bus.] Medium scale embedded systems have both hardware and software complexities. For complex software design, the following programming tools are available: C/C++/Visual C++/Java, RTOS, source code engineering tool, simulator, debugger and an integrated development environment. Software tools also provide solutions to hardware complexities.
3. ***Sophisticated embedded systems:*** Sophisticated embedded systems have enormous hardware and software complexities and may need several IPs, ASIPs, scalable processors or configurable processors and programmable logic arrays. They are used for cutting edge applications that need hardware and software co-design and components that have to be integrated in the final system. They are constrained by the processing speeds available in their hardware units. Certain software functions such as encryption

and deciphering algorithms, discrete cosine transformation and inverse transformation algorithms, TCP/IP protocol stacking and network driver functions are implemented in the hardware to obtain additional speeds. The software implements some of the functions of the hardware resources in the system. Development tools for these systems may not be readily available at a reasonable cost or may not be available at all. In some cases, a compiler or retargetable compiler might have to be developed for these. [A retargetable compiler is one that configures according to the given target configuration in a system.]

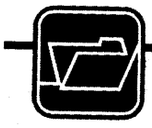
1.12 SKILLS REQUIRED FOR AN EMBEDDED SYSTEM DESIGNER

An embedded system designer has to develop a product using the available tools within the given specifications, cost and time frame. [Chapters 6, 13 and 14 will cover the design aspects of embedded systems.]

1. ***Skills for Small Scale Embedded System Designer:*** Author Tim Wilmshurst in his book states that the following skills are needed in the individua::::""""l or team that is developing a small-scale system: "Full understanding of microcontrollers with a basic knowledge of computer architecture, digital electronic design, software engineering, data communication, control engineering, motors and actuators, sensors and measurements, analog electronic design and IC design and manufacture". Specific skills will be needed in specific situations. For example, control engineering knowledge will be needed for design of control systems, and analog electronic design knowledge will be needed when designing the system interfaces. The basic aspects of the following topics will be described in this book to prepare the designer who already has a good knowledge of the microprocessor or microcontroller to be used. (i) Computer architecture and organization. (ii) Memories. (iii) Memory allocation. (iv) Interfacing memories. (v) Burning (a term used for porting) the executable machine codes in PROM or ROM. (v) Use of decoders and demultiplexers. (vi) Direct memory accesses. (vii) Ports. (viii) Device drivers in assembly. (ix) Simple and sophisticated buses. (x) Timers. (xi) Interrupt servicing mechanism. (xii) C programming elements. (xiii) Memory optimization. (xiv) Selection of hardware and microcontroller. (xv) Use of In-Circuit-Emulators (ICE), cross-assemblers and testing equipment. (xvi) Debugging the software and hardware bugs by using test vectors. Basic knowledge in other areas—software engineering, data communication, control engineering, motors and actuators, sensors and measurements, analog electronic design and IC design and manufacture—can be obtained from the standard text books available. A designer interested in small-scale embedded systems may not need at all concepts of interrupt latencies and deadlines and their handling, the RTOS programming tools described in Chapters 9 and 10 and the program models given in Chapter 6.
2. ***Skills for Medium Scale Embedded System Designer:*** Knowledge of 'C'/C++/Java programming, RTOS programming and program modeling skills are must to design medium-scale embedded-system. Knowledge of the following are critical. (i) Tasks or threads and their scheduling by RTOS. (ii) Cooperative and preemptive scheduling. (iii) Inter processor communication functions. (iv) Use of shared data, and programming the critical sections and re-entrant functions. (v) Use of semaphores, mailboxes, queues, sockets and pipes. (vi) Handling of interrupt-latencies and meeting task deadlines. (vii) Use of various RTOS functions. (viii) Use of physical and virtual device drivers. [Refer to Sections 4.2.6, 7.10 and 7.11.] Chapters 4 to 10 give detailed descriptions of these seven along with examples, and Chapters 11 and 12 provide on understanding of their use with the help of case studies. A designer must have access to an RTOS programming tool with Application Programming Interfaces (APIs) for the specific microcontroller to be used. Solutions to various functions like memory-allocation, timers, device drivers and interrupt handing mechanism are readily available as the APIs of the RTOS. The designer needs to

know only the hardware organization and use of these APIs. The microcontroller or processor then represents a small system element for the designer and a little knowledge may suffice.

3. ***Skills for Sophisticated Embedded System Designer:*** A team is needed to co-design and solve the high level complexities of hardware and software design. Embedded system hardware engineers should have skills in hardware units and basic knowledge of 'C'/C++ and Java, RTOS and other programming tools. Software engineers should have basic knowledge in hardware and a thorough knowledge of 'C', RTOS and other programming tools. A final optimum design solution is then obtained by system integration.



Summary

- An embedded system is one that has embedded software in a computer-hardware, which makes it a system dedicated for an application(s) or a specific part of an application or product or a part of a larger system.
- The embedded system processor can be a general-purpose processor chosen from a number of families of microprocessors. Alternatively, an ASIP for example microcontrollers, embedded processors and DSPs may be designed for specific application on a VLSI chip. An application specific instruction set processor (ASSP) may be additionally used for fast hardwired implementation of a certain part of the embedded software. A sophisticated embedded system may also use a multiprocessor or dual core unit.
- Embedded systems locate a software image in ROM. The image often consists of the following: (i) Boot up program. (ii) Initialization data. (iii) Strings for an initial screen display or system state. (iv) Programs for the multiple tasks that the system performs. (v) RTOS kernel.
- The embedded system needs a power source and controlled and optimized power dissipation from the total energy requirement. A charge pump provides a power-supply-less system in certain embedded systems.
- The embedded system needs clock and reset circuits. Use of the clock manager is a recent innovation.
- The embedded system needs interfaces: Input Output (IO) ports, serial UART and other ports to accept inputs and to send outputs by interacting with peripherals, display units, keypad or keyboard.
- The embedded system may need bus controllers for networking its buses with other systems.
- The embedded system needs timers and a watchdog timer for the system clock and for real-time program scheduling and control.
- The embedded system needs an interrupt-controlling unit.
- The embedded system may need an ADC for taking analog input from one or multiple sources. It needs a DAC using PWM for sending analog output to motors, speakers, sound systems, etc.
- The embedded system may need an LED or LCD or touch screen display units, keypad and keyboard, pulse dialer, modem, transmitter, multiplexers and demultiplexers.
- Embedded software is usually made in the high-level languages C, C++, Java or visual C++ with certain features added, enabled or disabled for programming. Use of 'C' and C++ also facilitates the incorporation of assembly language codes.
- The embedded system most often needs a real-time operating system for real-time programming and scheduling, device drivers, file system or device management and multitasking.
- The embedded system needs a debugger.
- A large number of applications and products employ embedded systems. A number of software tools are needed in the development and design phase of an embedded system.
- Five applications are described in detail: An automatic chocolate vending machine, a smart card, digital camera, mobile phone, mobile computer and robots playing an orchestra.
- A VLSI chip can embed ASIP or a GPP core and IPs for the specific application. A system on-chip is the concept in embedded systems; for example, a mobile phone in which analog and digital circuits, processors and software reside on a chip and become a system.

- The design process is abstracted by (i) definitions and analysis of system requirements, design metrics (Table 1.8) and validation requirements for finally developed systems specifications. There has to be consistency in the requirements, (ii) specifications (iii) architecture (iv) components (v) system integration.
- Design metrics are power dissipation, performance, process deadlines, user interfaces, size, engineering cost, manufacturing cost, flexibility, prototype development time, time-to-market, system and user safety and maintenance.
- The challenge in the process designing the system is to optimize competing design metrics.

Keywords and their Definitions

ADC : A circuit that converts the analog input to digital 8, 10 or 12 bits. The analog input is applied between positive and negative pins and is converted with respect to the reference voltage(s). When input equals reference positive and negative voltage, then all output bits equal 1, and when 0, then all output bits equal 0.

ASIP (Application Specific Instruction Set) : A processor with an instruction set designed for specific application on a VLSI chip; for example, microcontroller, DSP, IO, media, network or other domain-specific processor.

Assembler : A program that translates assembly language software into the machine codes placed in a file called '.exe' (executable) file.

ASSP (Application Specific System Processor) : A processing unit for system specific tasks, for example, image processing, compression and decompression, and that is integrated through the buses with the main processor in embedded system.

Bus : A bus consists of a common set of parallel lines to connect multiple devices, hardware units and systems for communication between two of these at any given instance. A communication protocol specifies the ways of communication of signals on bus. Protocol also specifies ways of arbitration when several devices need to communicate through the bus or ways of polling from the device requirements of the bus at an instance.

Cache : A fast read and write on-chip memory for the processor execution unit. It stores instructions and data fetched in advance from ROM or RAM for use in the execution unit and for data write back for RAM. It has an advantage in that the processor execution unit does not have to wait for instructions and data from external buses and also does faster write back of data meant for RAM.

Clock : Fixed frequency pulses that an oscillator circuit generates and that controls all operations during processing and all timing references of the system. Frequency depends on the needs of the processor circuit. A processor, if it needs a 100 MHz clock then its minimum instruction processing time is a reciprocal of it, which is 10 ns in a single cycle per instruction processing.

CODEC : A circuit for encoding the input information in fewer bits and for decoding the encoded information into the complete set of the original. Voice, speech, image and video signals and bits can be encoded and decoded. The CODEC also functions as a compression and decompression unit for voice, speech, image and video signals.

Coder : A coder which is a part of CODEC circuit is used to encode the input information.

Compiler : A program that, according to the processor specification, generates machine codes from high level language. The codes are called object codes.

Cycle : A cycle consists of fetch of an instruction from RAM/ROM, its execution at the processor and writing back the results of the operations.

DAC : Digital bits (8 or 10 or 12) converted to analog signal scaled to a reference voltage. When all input bits equal 1, then analog output equals the difference between the positive and negative reference pin voltages; when all input bits equal 0, then the analog output equals negative pin reference voltage.

Decoder : A decoder circuit that decodes the input address and activates a selected output among the many outputs. It is used for selecting one among several addressable units. A decoder also decodes the encoded bits and generates the output bits, signal or information.

Demultiplexer : A digital circuit that has digital outputs in multiple channels. The channel to which output is sent from the input is the one that is addressed by the channel address bits at the demultiplexer input. A demultiplexer takes the input and transfers it to a select channel output among the multiple output channels.

Design metrics : The parameters that define design requirements and must be kept in view during each stage of design process.

Device Driver : High level language functions, such as open (configure), connect, bind, listen, read or write or close the device. Each function calls an interrupt service routine (software) that runs after the programming of control register (or word) of a peripheral device (or virtual device) to allow the device inputs or outputs. The routine reads the status register for device status, gets the inputs from the device and writes the output to the device

Device Manager : Software to manage multiple devices and their drivers.

Device Programmer : It takes the inputs from a file generated by the locator and burns the fusable link to actually store the data and system codes at the ROM.

Embedded system : A system that has embedded software in a computer-hardware that makes it a system dedicated for an application or a specific part of an application or product, or a part of a larger system.

File : A data structure (or virtual device) that sends the records (characters or words) to a data sink (for example, a program) and that stores the data from the data source (for example, a program). A file in a computer may also be stored in the hard disk.

File System : A file system specifies the ways a file can be created, called, named, used, copied, saved or deleted.

FPGA : These are Field Programmable Gate Arrays on a chip. The chip has a large number of arrays with each element having links. Each element of array consists of several XOR, AND, OR, multiplexer, demultiplexer and tristate gates. Complex digital circuit functions are created by appropriate programming of the links on transferring data from memory.

GPP (General-purpose processor) : A processor from a number of families of processors, microcontrollers, embedded processors and digital signal processors (DSPs) having a general-purpose instruction set and readily available compilers to enable programming in a high level language.

Input Output (IO) ports : The system gets the inputs and outputs from these. Through these, the keypad or LCD units or touchscreen or peripherals and external systems connect to the system.

Interrupt controller (handler) : A unit that controls (handles) processor operations arising out of an interrupt from a source.

Kernel : An OS program with the following functions: memory allocation and deallocation, task scheduling, interprocess communication, effective management of shared memory access by using signals, exception (error) handling signals, semaphores, queues, mailboxes, pipes and sockets I/O management, file system, interrupts control (handler), device drivers and device management.

LCD : Liquid crystal display—a crystal that absorbs or emits light on application of 3 to 4 V 50 or 60 Hz voltage pulses with currents ~ 50 μA. Multisegment and multiline LCD units are used for a display of digits, characters, charts, pictograms and short messages with very low power dissipation.

LED : Light Emitting diode—a diode that emits red, green, yellow or infrared light on forward biasing between 1.6 V to 2 V and currents between 8–15 mA. Multisegment and multiline LED units are used for bright display of digits, characters, charts and short messages.

Linker : A program that links the compiled codes with other codes and provides input for a loader or locator.

Loader : A program that reallocates the physical memory addresses for loading into the system RAM memory. Reallocation is necessary, as the available memory may not start from 0x0000 at any given instant of processing in a computer. The loader is a part of the OS in a computer.

Locator : A program to reallocate the linked files of the program application and the RTOS codes at the actual addresses of the ROM memory. It creates a file in a standard format. The file is called a ROM image.

Mask and ROM mask : Created at a foundry for fabrication of a chip. The ROM mask is created from a ROM image file.

Memory : This stores all the programs, input data and output data. The processor fetches instructions from it and gives the processed results back to it.

Memory Stick : A memory stick (card) is unit of memory for video, images, songs, or speeches and is used as large storage in digital camera and mobile systems. For example, Sony memory stick Micro (M2) has size 15×12.5×1.2 mm^3 and functions as flash memory of 2 GB.]

Multiplexer : A digital circuit that has digital inputs at any instance in multiple channels. The channel for which the output is sent from the input is the one that is addressed by the channel address bits at the multiplexer input. A multiplexer takes the input among the multiple input channels and transfers a selected channel input to the output channel.

Microcontroller : A unit with a processor. Memory, timers, a watchdog timer, interrupt controller, ADC or PWM, and so on are provided as required by the application.

Modem : A circuit to modulate the outgoing bits into pulses usually used on the telephone line and to demodulate the incoming pulses into bits for incoming messages.

Multitasking : Processing codes for the different tasks as directed by the OS.

Physical Device : A device like a printer or keypad connected to the system port.

Pipe : A data structure (or virtual device), which is sent a byte stream from a data source (for example, a program) and which delivers the byte stream to a data sink (for example, a printer).

Process : A program or task or thread that has a distinct memory allocation of its own and has one or more functions or procedures for a specific job. The process may share

the memory (data) with other tasks. A processor may run multiple processes separately or concurrently as directed by the OS.

Processor : A processor executes the instruction cycles and executes one process or many as per the command (instruction) given to it.

Pulse Width Modulator (PWM) : A modulator that modulates the pulse width as per the input bits. It provides a pulse of width scaled to the analog output desired. On integrating PWM output the desired DAC operation is achieved.

Real-time operating system : Operating system software for real-time programming and scheduling, process and memory manager, device drivers, device management and multitasking.

RAM : This is a random access read and write memory that the processor uses to store programs and data that are volatile and which disappear on power down or when switched off.

Registers : These are associated with the processor and temporarily store the variable values from the memory and from the execution unit during processing of an instruction.

Reset : A processor state in which the processor registers acquire initial values and from which starts an initial program; this program is usually the one that also runs on power up.

Reset circuit : A circuit to force a reset state that gets activated for a short period on power up. When reset is activated, the processor generates a reset signal for the other system units needing reset.

ROM : A read only memory that locates the following in it—embedded software, initial data and strings and operating system or RTOS.

System : A way of working, organizing or doing one or a series of tasks by following a fixed plan, program and set of rules.

System on Chip : A system on a VLSI chip that has all the necessary needed analog as well as digital circuits; for example, in a mobile phone.

Timer : A unit to provide the time for the system clock and real-time operations and scheduling. It generates interrupts on timeouts as per the preset time or on overflow or on successful comparison of present time with a preset time or on capturing the time on an event.

Touchscreen : An input as well as an output device that is used to enter a command, choose a menu or to give user reply as input by physically touching at a screen position either by the finger or by a stylus. A stylus is thin pencil-shaped object. It is held between the fingers and used just as a pen is used to mark a dot. The screen displays the choices or commands, menus, dialog boxes and icons, similar to a computer display.

UART : Universal Asynchronous Receiver and Transmitter.

Virtual Device : A file or socket or pipe-like device that is programmable for opening, closing, reading and writing similar to a physical device.

VLSI chip : A very large-scale integrated circuit made on silicon with ~1M and above transistors.

Watchdog timer : A timer that resets the processor in case the program gets stuck for an unexpected length of time.

Review Questions

1. Define a system. Now define an embedded system.
2. What are the essential structural units in the following? (a) a microprocessor (b) an embedded processor (c) a microcontroller (d) a DSP (e) an ASIP. List each of these.
3. How does a DSP differ from a general-purpose processor (GPP)? [Sections 1.2.1, 1.7.3 and 2.3.3].
4. What are the advantages and disadvantages of the following? (a) a processor with only a fixed-point arithmetic unit and (b) a processor with additional floating-point arithmetic processing unit.
5. How does a microcontroller differ from a DSP? [Sections 1.2.3, 1.7.2, 2.1 and 2.3.5].
6. Explain single purpose processors use in convergence technology embedded systems (a) smart mobile phone with mail client, Internet connectivity and image-frame downloads and (b) digital camera.
7. Compare features in a family chip (or core) of each of the following: a microprocessor, microcontroller, RISC processor, DSP and ASSP.
8. Why do later generation systems operate the processor at low voltages (≤ 2 V) and perform IOs at (~3.3 V)?
9. What are the techniques of power and energy management in a system?
10. What is the advantage of running a processor at reduced clock speed in certain sections of instructions and at full speed in other sections?
11. What is the advantage of the following? (a) Stop instruction (b) Wait instruction (c) Processor idle mode operation (d) Cache-use disable instruction (e) Cache with multiways and blocks in an embedded system.
12. What do we mean by charge pump? How does a charge pump supply power in an embedded system without using power-supply lines?
13. What do you mean by 'real time' and 'real time clock'?
14. What is the role of processor reset and system reset?
15. Explain the need of watchdog timer and reset after the watched time.
16. What is the role of RAM in an embedded system?
17. Why do we need multiple actions and multiple controlling tasks for devices in an embedded system? Explain, using as an example the embedded system, the remote of colour TV.
18. When do we need multitasking OS?
19. When do we need an RTOS?
20. Why should the embedded system RTOS be scalable?
21. Explain the terms IP core, FPGA, CPLD, PLA and PAL
22. What do you mean by System-on-Chip (SoC)? How will the definition of an embedded system change with a System-on-Chip?
23. What are the advantages offered by an FPGA for designing an embedded system?
24. What are the advantages offered by an ASIC for designing an embedded system?
25. What are the advantages offered by an ASIP for designing an embedded system?
26. Real time video processing needs sophisticated embedded systems with hard real time constraints. Why? Explain.
27. Why does a processor system always need an 'Interrupts Handler (Interrupt Controller)'?
28. What role does a linker play?
29. Why do we use a loader in a computer system and a locator in an embedded system?
30. Why does a program reside in the ROM in the embedded system?
31. Define ROM image and explain each section of an ROM image in a system.
32. When is the compressed program and data in ROM used? Give five examples of embedded systems having these in their ROM images.
33. When is SRAM used and when DRAM? Explain your answers.
34. What do we mean by the following: physical device, virtual device, plug and play device, bus self-powered device, device management and device-specific processor.

35. Define design metrics in embedded systems. What are the different competing design metrics? What are the constraints of embedded system design?
36. How is power dissipation optimized?
37. What are the challenges faced in designing an embedded system?

Practice Exercises

38. Search for the definitions of embedded system in the books referred to in the 'References' section and tabulate these with the definitions in column 1 and the reference names in column 2.
39. Classify the embedded systems into small scale, medium scale and sophisticated systems. Now, reclassify these embedded systems with and without real-time (response time constrained) systems and give 10 examples of each.
40. An automobile cruise control system is to be designed in a project. What will be skills needed in the team of hardware and software engineers?
41. Take a value, x = 1.7320508075688. It is squared once again by a floating-point arithmetic processor unit. Now x is squared by a 16-bit integer fixed point arithmetic processing unit. How does the result differ? [*Note*: Fixed-point unit will multiply only 17320 with 17320, divide the result by 10000 and then again divide the result by 10000.]
42. Design a four-column table. Write two examples of embedded systems in columns 2 and 3. In Column 1, write the type of processor needed among the following: microprocessor, microcontroller, embedded processor, digital signal processor, ASSP, single purpose and media processor. Give your reasoning in column 4.
43. Why does a CMOS IO circuit power dissipation reduce by compared to 5 V, factor of half, ~$(3.3/5)^2$, in IO 3.3 V operation?
44. What will be the reduction in power dissipation for a CMOS circuit when voltage reduces from a 5 V to a 1.8 V operation?
45. List the various type of memories and application of each in the following: robot, electronic smart weight display system, ECG LCD display-cum-recorder, router, digital camera, speech processing, smart card, embedded firewall/ router, mail client card, and transreceiver system with collision control and jabber control. [Collision control means transmission and reception when no other system on the network is using the network. Jabber control means control of continuous streams of random data flowing on a network, which eventually chokes a network.]
46. Tabulate hardware units needed in each of the systems: camera, mobile computer and robot.
47. Give two examples of embedded systems, which need one or more of following units. (a) DAC (Using a PWM) (b) ADC (c) LCD display (d) LED displays (e) Keypad.(f) Pulse dialer (g) Modem (h) Transreceiver.
48. An ADC is a 10-bit ADC. It has reference voltages, $V_{ref-} = 0.0$ V and $V_{ref+} = 1.023$ V. What will be the ADC outputs when inputs are (a) –0.512 V (b) +0.512 V and (c) +2047V? What will be the ADC outputs in three situations when (i) $V_{ref-} = 0.512$ V and $V_{ref+} = 1.023$ V (ii) $V_{ref-} = 1.024$ V and $V_{ref+} = 2.047$ V and (ii) $V_{ref-} = -1.024$ V and $V_{ref+} = +2.047$ V.
49. Tabulate the advantages and disadvantages of using coding languages as follows: (a) Machine coding (b) Assembly (c) C (d) C++ and (e) Java.
50. List the software tools needed in designing each of the embedded system—camera, mobile phone and robot.
51. Justify the use of physical and virtual devices drivers in embedded systems.
52. The cost of designing an embedded system may be thousands of times the cost of its processor and hardware units. Explain this statement.
53. An FPGA (Field Programmable Gate Arrays) core integrated with a single or multiple processor unit on chip. How do these help in the design of sophisticated embedded systems for real time video processing?
54. List the memory units and processor needed in a smart card.

8051 and Advanced Processor Architectures, Memory Organization and Real-world Interfacing

2

Recall

The previous chapter dealt with the following:

- Embedded system embeds software and RTOS.
- Embedded system embeds software in hardware consisting of microprocessor or microcontroller or DSP or ASIP and single purpose processors.
- Embedded system has memory (ROM, RAM and caches), ports, timers, devices and interfacing circuits.
- Microcontroller hardware consists of processing unit, RAM, ROM, timing and interrupt handling devices and other application specific units as a single chip or VLSI core.
- Design metrics, processes and challenges.
- Software, device drivers and device-manager.
- Software tools.

LEARNING OBJECTIVES

In this chapter, we will learn the following

1. *8051 architecture in brief and its processor, memory, ports, counters/timers, serial IO and interrupt handler units*
2. *Real world interfacing, and internal and external buses that interconnect the processor with the system memories, IO devices and all other system units*
3. *Interfacing examples with keyboard, displays, ADC and DACs*
4. *Advanced processors x86, ARM and SHARC architectures*
5. *Processor and memory organization*
6. *Instruction-level parallelism and superscalar, processing, pipelining and cache units for improved computational performance of the processor by faster program execution*
7. *Various types of memory and their uses*
8. *Devices and memory addresses allocations*
9. *Performance metrics to measure the performance of a processor*
10. *Processor selection for embedded system*
11. *Memory selection for embedded system*

2.1 8051 ARCHITECTURE

The following subsections summarize the 8051 architecture in brief. A reader may refer to a standard text for details.

2.1.1 8051 Microcontroller Architecture

Figure 2.1 shows the architecture of the classic 8051 microcontroller. Classic means the original version, based upon which new enhancements and versions are provided. The classic version consists of following hardware:

1. A 12 MHz clock. Processor instruction cycle time is 1 μs.
2. An 8-bit ALU. The internal bus width is 8-bit.
3. CISC (Complex Instruction Set Computer) architecture. [CISC provides many modes for addressing operands in arithmetic, logical and other instructions. Several complex instructions take more than one cycle time. Complex instructions implement in hardware not by separate hardwired logic circuits for each instruction but by a microprogram control circuit.]
4. Special bit manipulation instructions.
5. A program counter, in which the initial default reset value defined by the processor is 0x0000.
6. A stack pointer, in which the initial default value defined by the processor is 0x07.

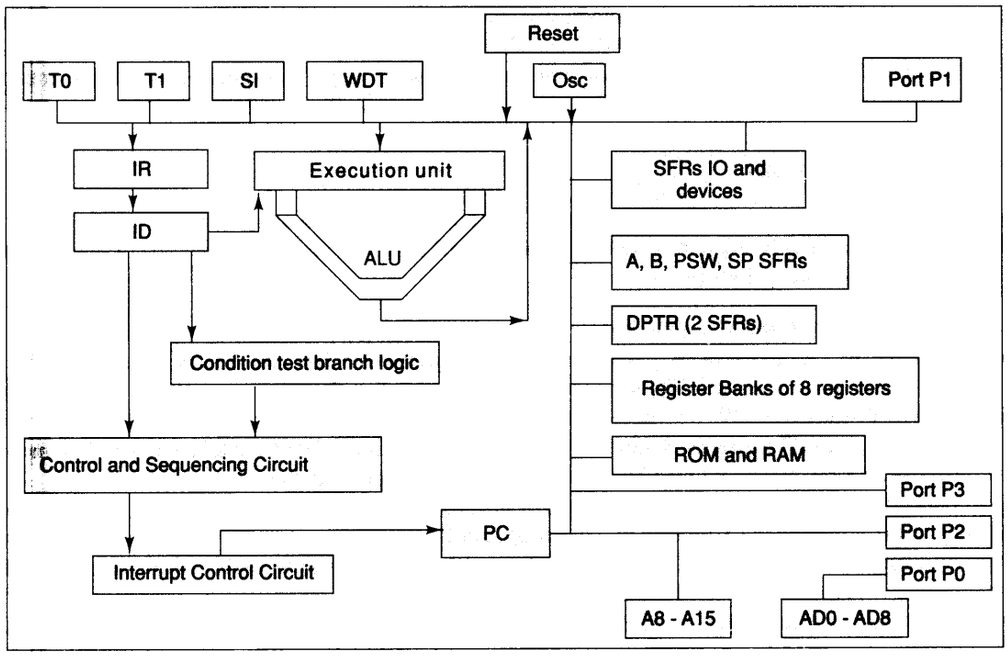


Fig. 2.1 8051 Architecture

7. A simple architecture, with no floating-point processor, no cache, no memory management unit, no atomic operations unit, no pipeline and no instruction level parallelism. (Sections 2.3 and 2.5). There is no DMA controller (Section 4.8) in the classic and most other versions.
8. A Harvard memory architecture (Section 2.4.2). The program memory and data memory have separate address spaces from 0x0000 and separate control signal(s).
9. On-chip RAM of 128 bytes. The 8052 version provides for RAM of 256 bytes; 32 bytes of RAM are also used as four banks (sets) of registers. Each register-set (bank) thus has eight registers. The external data/stack memory can be added upto 64 kB in most versions. In certain 8051 enhancements, this limit has been enhanced to 16 MB.
10. There are special function registers (SFRs). These are PSW (processor status word), A (accumulator), B register, SP (stack pointer) and registers for serial IOs, timers, ports and interrupt handler.
11. 8351 version has on-chip ROM; 8751 version EPROM; 8951 version has on-chip EEPROM or flash memory of 4 kB. Several versions provide for higher capacity ROM. Additional program memory can be added externally upto 64 kB. In extended 8051 and unified address space versions (8051 EX and MX versions), this limit has been extended to 16 MB.
12. Two external interrupt pins, INT0 and INT1.
13. Four ports P0, P1, P2 and P3 of 8 bits each in single chip mode. (Section 2.1.3) There are two timers (Section 2.1.5) and a serial interface (SI). It can be programmed for three full duplex UART modes for a serial IO. [IO with each bit of a word successive transmitted on the data line for a time interval.] The SI can also be programmed for half duplex synchronous IO (Section 2.1.6).

14. Classic version has no pulse width modulator and provides on support to DAC. (Section 1.3.7) It has no modem, no watchdog timer, no ADC. Certain versions support watchdog timer and ADC. Siemens SAB 80535-N supports ADC with programmable reference voltage. Advanced versions support these features and choice of version depends on system requirement. (Section 2.8 and 2.9).

2.1.2 Instruction Set

Figure 2.2 shows instruction types in the 8051 set. There are seven types of instructions.

Full instruction set and instructions in detail can be referred to from a microcontroller text or manual. The important instructions are as follows:

Data Transfer Instructions Data transfer instructions move (copy) one source operand to another destination. `MOV A, Rn` and `MOV Rn, A` are for moving (copying) the data into A from R_n and to R_n. R_n is the n^{th} register in a set of 8 registers. PSW bits RS0 and RS1 predefine the set.

Data transfer instructions `MOV A, @Ri` and `MOV @Ri, A` are for moving (copying) the data into A from `@Ri` and to `@Ri`. R_i is the i^{th} register in a set of 8 registers. PSW bits RS0 and RS1 predefine the set. @R_i means data transfer to an 8-bit address pointed by the contents of the i^{th} register in the set.

Four data transfer instructions are `MOV direct,#data, MOV A,#data, MOV Rn,#data` and `MOV @Ri,#data` for moving 8-bit data into direct or A or R_n or `@Ri` Direct means data transfer to an 8-bit address of internal 128B RAM or SFR address. @R_i means data transfer to an 8-bit address pointed by the contents of the i^{th} register in the set (i = 0 or i).

Seven data transfer instructions are `MOV direct,direct, MOV A,direct, MOV direct, A, MOV direct,Rn, MOV Rn,direct, MOV direct,@Ri,` and `MOV @Ri, direct` are for moving data between the 8-bit direct address to direct or A or R_n or `@Ri`. Direct means data transfer to or from an 8-bit address of internal 128B RAM or SFR address. @R_i means data transfer to an 8-bit address pointed by contents of i^{th} register in the set (i = 0 or 1).

There is a 16-bit external memory data pointer, DPTR. The `MOV DPTR, data16` instruction is used to send 16 bits specified in data16 to DPTR.

MOVX (move external instruction) will transfer the data to or from external data memory. These instructions are `MOVX A, @DPTR` and `MOVX @DPTR, A`. @DPTR means address as pointed by 16 bits of DPTR. For an 8-bit external memory address, instead of DPTR, @R_i is used (i = 0 or 1). MOVC (move code from external program memory instruction) will transfer the data from the external program memory. Instructions are `MOVC A, @A + DPTR` and `MOVC A, @A + PC`.

For stack operations, there are `PUSH direct` and `POP direct` instructions.

Bit Manipulation Instructions Each bit of certain SFRs and an 8-byte internal RAM are assigned bit addresses in 8051 hardware. There are bit manipulation instructions to clear, set, AND, OR, MOV the bit.

Byte Manipulation Instructions There are byte manipulation instructions to rotate right A, rotate left A, rotate right A with carry, rotate left A with carry, complement A, clear A and swap with A lower and upper nibble (set of 4 bits).

Arithmetic Instructions Arithmetic instructions of the source operand is stored in the accumulator and the result of the operation is also stored in the accumulator. For example, `ADD A, Rn`. It adds the byte in A with the byte in the n^{th} register. ($A \leftarrow A + R_n$). Three arithmetic instructions are ADD, ADDC (add including carry bit) and SBBB (subtract including borrow bit). Carry bit is set to 1 when an operation results in carry or

borrow. Borrow is saved in the carry bit. The second operand can be R_n, direct, @R_i or #data (i = 0 or 1). The meaning of these is the same as in case of data transferred instructions (explained earlier). There is also an instruction to adjust hexadecimal addition to decimal addition.

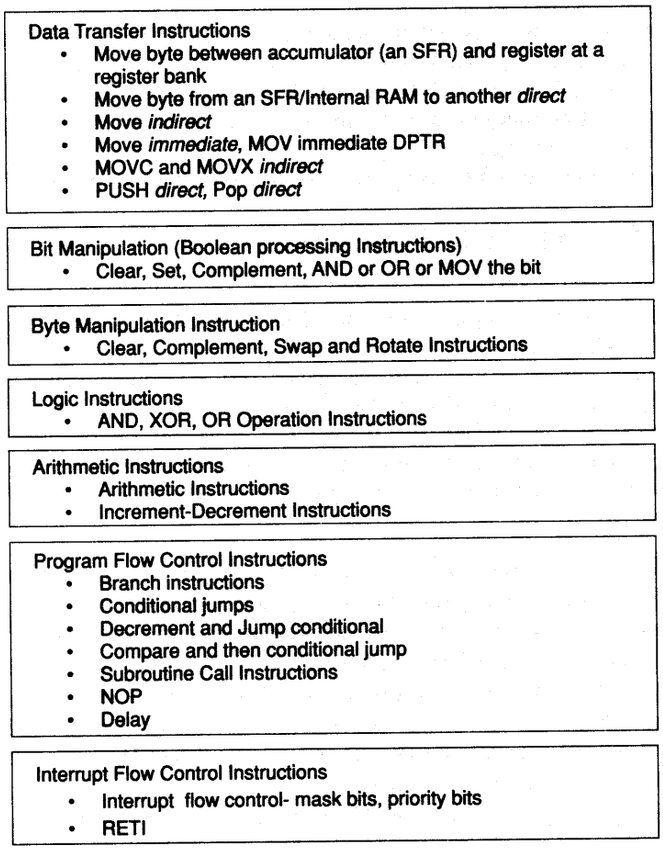


Fig. 2.2 Instruction types in 8051 instruction set

INC and DEC instructions increase (by 1) and decrease (by 1) the bits at the source. The source can be R_n, direct, @R_i or A.

`MUL AB` and `DIV AB` are used to find A – B ← A × B and A – B ← A ÷ B. The multiplication results in lower byte in A and higher in B. Division results quotient is stored in A and the remainder in B.

Logic Instructions Logic instructions one of the source operand is accumulator or direct and result of operation is also in the accumulator or direct. For example, `ANL A,` R_n and `ANL direct,` R_n. It logical ANDs the byte in A or direct with the byte in n^{th} register in the register set. (A ← A . R_n). Three logic instructions are ANL (AND logic), ORL (OR logic) and XRL (XOR logic). The second source operand is @R_i or R_n or direct when the first source cum destination is A, and is A and #data when the first source cum destination is *direct*. [Meaning of these are the same as in the case of the data transfer instructions (explained earlier).]

Program Flow Control Instructions Program flow control is done by instructions for short jump, absolute jump or long jump or jump. Short jump instruction is to a relative address within –128 and +127. Absolute jump instruction is to direct 11-bit address in program memory. Long jump instruction is to direct 16-bit address in program memory. Jump instruction is pointed by A + @DPTR.

Conditional program flow control instructions are also present. Loop control instructions are also present in which jump count value is in R_n or direct, and offset. Both are specified in the loop control instruction.

Program flow control in a subroutine call is by absolute call or long call instruction. Absolute call instruction is to call a specified direct 11-bit address at program memory. Long call instruction is to call a specified direct 16-bit address directly at program memory.

Return from a called routine is by RET instruction and return from interrupt service routine is by RETI.

2.1.3 IO Ports, Circuits and IO Programming

Figure 2.3(a) shows P0, P1, P2 and P3 IO ports in 8051. 8051 in single chip mode has four ports. Single chip means there are no external memory chips or ports or serial port or peripheral attached to the port. Section 2.1.4 will give an expanded mode circuit.

Port driving capabilities depend on the specific version of 8051. P0 is an 8-bit open drain bidirectional IO port and P1 to P3 are quasi bidirectional IO ports. Open drain port means that the output port pins need a pull up circuit or resistance to raise the voltage level to logic 1. A quasi bidirectional IO port can drive for two clock cycles eight logic LSTTL gates in 8051. For higher driving capability, pull up circuits will be required. Port P1 bits are open drain in P83C538 version as two bits P1.6 and P1.7 are used for I^2C bus (Section 3.10.1) clock and data signals.

IO Port Circuits Sections 2.2.6 and 3.3 will describe the interfacing circuit of port IO bits to switches, keypad, encoders, motors and LCD controllers. Figures 2.3(b) and (c) show IO port P1 circuits for two stepper motors in a printer and six servomotors in a robot. IO port bytes and bits are programmed and accessed as follows:

(i) *IO Byte Programming* The internal IO ports P0, P1, P2 and P3 in the 8051 have byte addresses to access and perform read, write or other operations. These addresses are the direct 8-bit addresses of each that are specified in the instructions. Addresses of bytes at P0, P1, P2 and P3 have 0x80, 0x90, 0xA0 and 0xB0. All instructions in the instruction set using *direct* addresses can be used to access and perform read or write operations on the ports.

(ii) *IO Port Bit Programming* Each port P0, P1, P2 and P3 has 8 bits and each bit has addresses to access and perform read or write operations using bit-manipulation instructions. These addresses are the *bit* address of 8 bits, which are specified in the instructions. Bits P0.0 to P0.7 have addresses 0x80 to 0x87. Bits P0.0 to P0.7 have addresses 0x80 to 0x87. Bits P1.0 to P1.7 have addresses 0x90 to 0x97, P2.0 to P2.7, 0xA0 to 0xA7 and P3.0 to P3.7 0xB0 to 0xB7. All instructions in the instruction set using *bit* addresses can be used to access and perform complement or read or write operations. The C flag in PSW is the accumulator for logic operations on the bits using bit-addresses.

Example 2.1

1. `MOV 0xA0, #0xFF` will move bits to port P2 and P2 bits will become = 11111111_b.
2. `MOV 0x90, #0x1C` will move bits at port P1 = 00011100_b. After this instruction, `INC 0x90` will make P1 = $00011100_b + 1 = 00011101_b$.

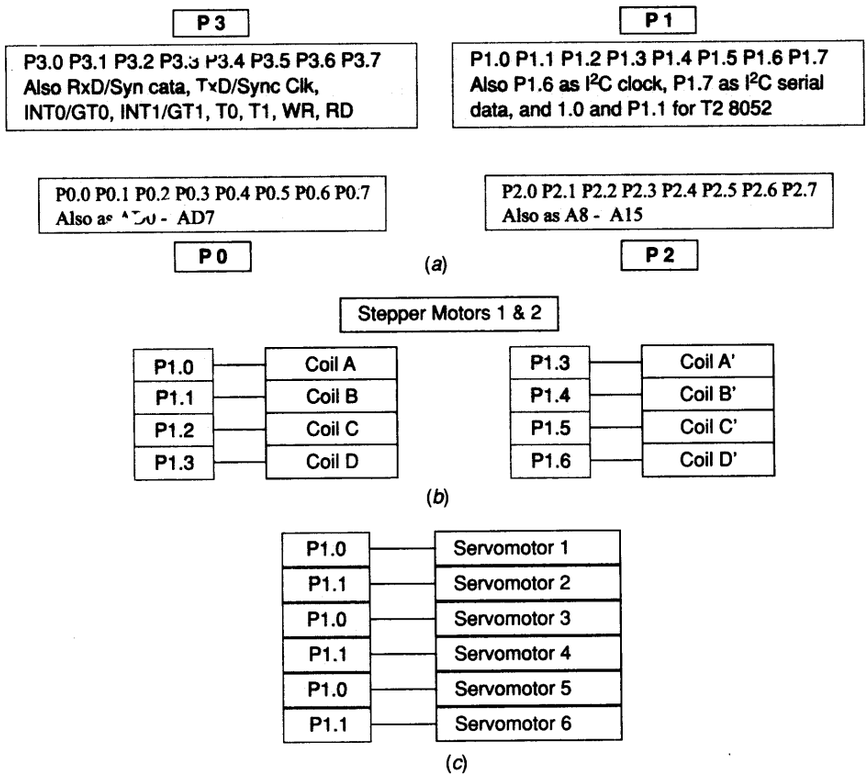


Fig. 2.3 (*a*) IO ports in 8051 (*b*) IO port P1 circuit for two stepper motors in a printer (*c*) IO port circuit for six servo motors in a robot

Example 2.2

1. `CPL 0x90` will complement the bit 0 at port P1.
2. `CLR 0x80` will make P0.0 as 0. Now after a delay of period= T1, the `SETB 0x80` will make P0.0 as 1. Now after a delay of period = T2, the `CLRB 0x80` will make gain P0.0 as 0. A pulse of time-period T2 and duty cycle 100 T1/(T1 + T2) creates if the instructions are executed in a loop.
3. `SETB C` will set carry bit in PSW to 1. After this operation, `ANL C, 0x93` will perform logic AND operation between bits C and P1.3 and result will be in C. If P1.3 = 0 then C will become 0 else C will remain 1.
4. `CLR C` will reset (clear) carry bit in PSW to 0. After the operation, `ORL C, 0xB2` instruction will perform logic OR operation between bits C and P3.2 and result will be in C. If P3.2 = 0 then C will remain 0 else C will remain 1. After the operation, `MOV 0x85, C` instruction will move result in C to P0.5.

2.1.4 External Memory Interfacing Circuits

Figure 2.4(a) shows how to connect the external program and data memory circuits in 8051. There are two sets of memory, program memory and data memory. The processor has two control signals $\overline{PSEN}$ and $\overline{RD}$ to control read from program or data memory. The processor has a control signal ALE to control use of AD0-AD7 as address or data at a given instance. Section 2.2.1 will explain PSEN, RD and ALE control signals.

1. Port P0 is used in expanded mode as AD0-AD7. AD0-AD7 are the multiplexed signals of the A0-A7 lower address bits of the address bus and D0-D7 bits of data bus. A0-A7 and D0-D7 are time division multiplexed. For an interval the processor activates $\overline{ALE}$ (address latch enable) in an instruction cycle and the AD0-AD7 lines have A0-A7; a latch-circuit separates A0-A7 signals for the memory.
2. Port P2 has A8-A15 address signals. When the processor activates $\overline{PSEN}$ (Program store enable), it reads the byte from the external program memory through the D0-D7 data bus. When the processor activates $\overline{RD}$ (read), it reads the byte from the external data memory through the D0-D7 data bus.

For addresses outside the internal RAM, SFR and internal program memory, the processor always accesses the external memory. That is irrespective of external enable EA active or inactive. Internal RAM and SFR addresses between 0x00 and 0xFF are the same as external memory addresses 0x0000 and 0xFFFF. Internal program memory addresses between 0x0000 to 0xFFF (in case of 4 kB internal ROM) are the same as the external program memory addresses 0x0000 and 0xFFFF. When a control signal EA activates, the processor accesses the external addresses in the memory instead of these internal memory or register addresses.

The 8051 has a memory mapped IO (Section 2.2.2). Memory and ports are assigned addresses such that each has a distinct range of addresses in the data memory address space. Therefore, interfacing circuit design is identical to that for the memory and connects the external ports and programmable peripheral interface (PPI). Memory and ports are assigned the addresses such that each has a distinct range of addresses. A PPI chip is 8255. Figure 2.4(b) shows the interfacing when using the external PPI ports PA, PB and PC.

2.1.5 Counters and Timers

Figure 2.5 shows the specifications of counters and timers, T0 and T1 in 8051. There are two timing and counting devices T0 and T1 in classic 8051 and three T0, T1 and T2 in 8052. Using the two SFRs, TH1-TL1, the counts at the higher and lower 8 bits of T1 are accessed. The SFRs hold the T1 device 16 bits. Using two SFRs, TH0-TL0, the higher and lower 8 bits of T0 are accessed. The SFRs hold the T0 device 16 bits.

An SFR called TMOD controls the T1 and T0 modes using the upper and lower 4 bits each, which programs the counting/timing of T1 and T0. A bit for each controls whether the external gate input controls or not. Another bit controls whether counter or timer mode is used. Two other bits control the functional mode of timer/counter as mode 0 or 1 or 2.

The counting/timing device records time when inputs are given by the clock. The clock pulses are internally given at the specific time intervals in case of functioning as timer. It also records counts when the inputs are given externally. The counter is given the input to count from external input pins.

1. When timing or counting devices are externally controlled by the gate inputs, when GT0 or GT1 is externally activated, the device can function; else, it deactivates in gate input mode. GT0 or GT1 signals are given through the P3.2 and P3.3.
2. When T0 is in counter mode, it is given the input to count from the external input pin T0 at P3.4. When T1 is in counter mode, it is given the input to count from the external input pin T1 at P3.5.

The upper four 4 bits of an SFR, called the TCON, programs the counting/timing device in T1 and T0 modes. TCON.7 and TCON.5 show the timer/counter overflow status for T1 and T0, respectively. TCON.6

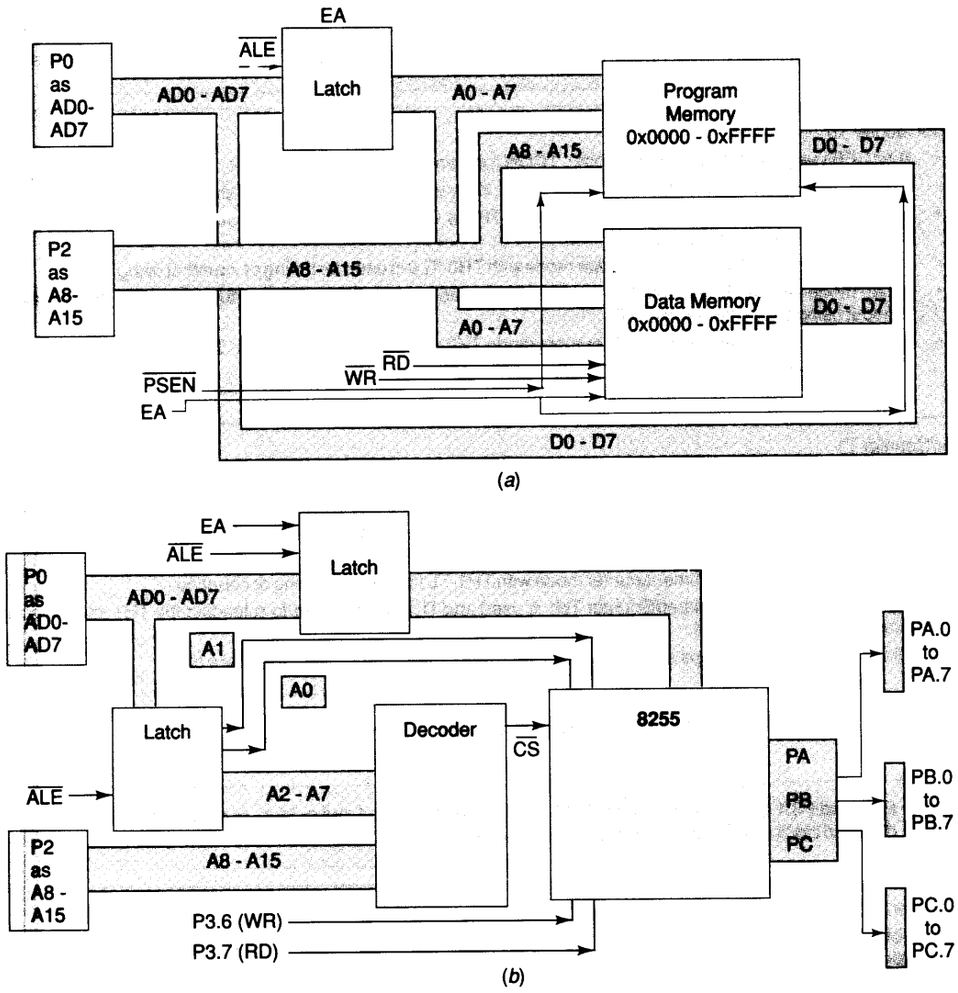


Fig. 2.4 (*a*) Connection of 8051 to external program and data memory circuits (*b*) Interfacing of 8051 to external PPI 8255 ports PA, PB and PC

and TCON.4 control the start and stop of the timer/counter overflow status for T1 and T0, respectively. The lower bits of the TCON are used for interrupt control for INT0 and INT1 (Section 2.1.7).

2.1.6 Serial Data Communication Input/Output

Figure 2.6 shows serial ports and data serial communication using SI (serial interface) in 8051.

There are two SI modes: half duplex synchronous and full-duplex asynchronous. Half duplex means one-way communication and full duplex means both ways at the same instance.

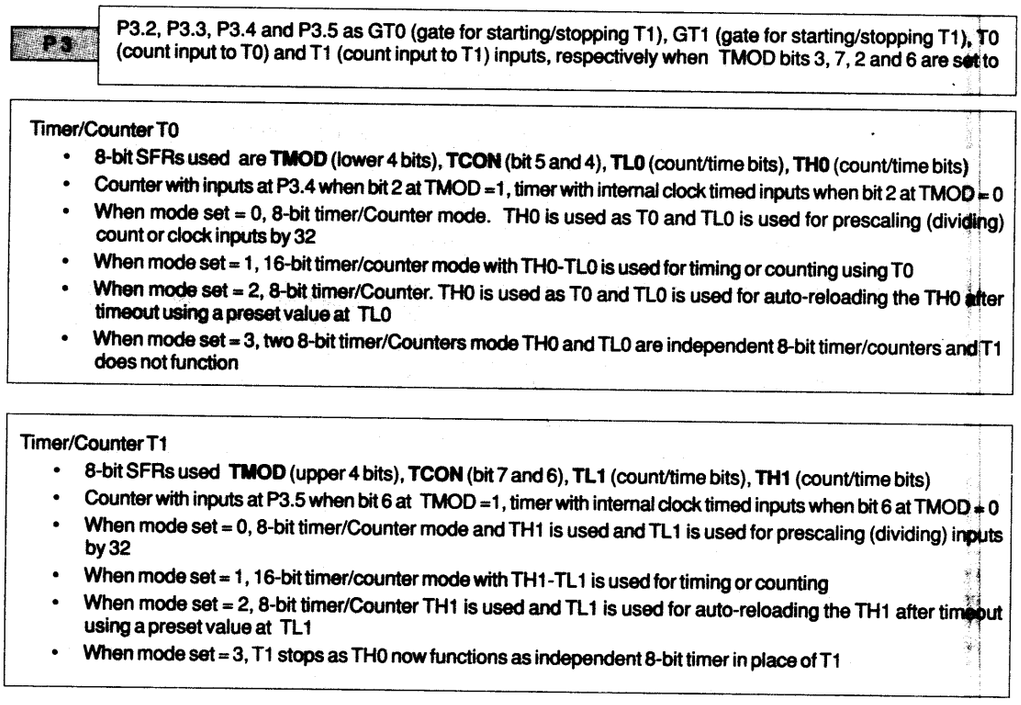


Fig. 2.5 Counter-cum-timers T0 and T1 in 8051

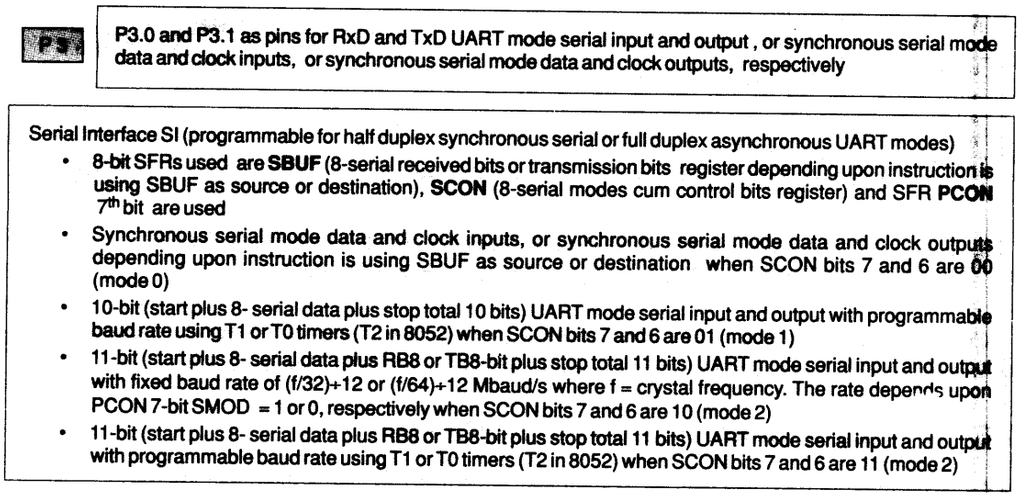


Fig. 2.6 Serial ports and data serial communication using SI (serial interface) in 8051

Using the single SFR for transmit and receive byte buffers, the serial output or input is sent. The SFRs hold the SI transmission 8 bits when it is written. 0x99 is the address of SI buffers. For example, `MOV 0x99, A` instruction *writes* into transmission buffer from the *A* register and `MOV R1, 0x99` instruction reads the R1 register from the receive buffer.

An SFR called SCON controls the SI interface. Three upper bits program the modes as 0, 1, 2 or 3. Mode 0 is half duplex synchronous. Modes 1 or 2 or 3 are full duplex asynchronous. Mode 2 transmits and receives in 11 T format and mode 1 in 10T format. T is the interval between successive transmitted or received bits and T^{-1} is the baud rate. (Section 3.2.3 gives the details). Modes 1 and 3 are for programmable baud rate and 2 for fixed baud rate.

A bit SCON.4 enables or disables SI receiver functions. Two bits SCON.3 and SCON.2 specify the 8^{th} bit to be transmitted and 8^{th} bit received when the mode is 2 or 3. A bit SCON.1 enables or disables SI transmitter interrupts (TI) on completion of transmission. A bit SCON.0 enables or disables SI receiver interrupts (RI) on completion of transmission.

2.1.7 Interrupts in 8051

Figure 2.7 shows the specifications of system of interrupts in classic 8051. There are multiple interrupts in 8051. When an interrupt is enabled (not masked), then on occurrence of that interrupt event, an ISR is called.

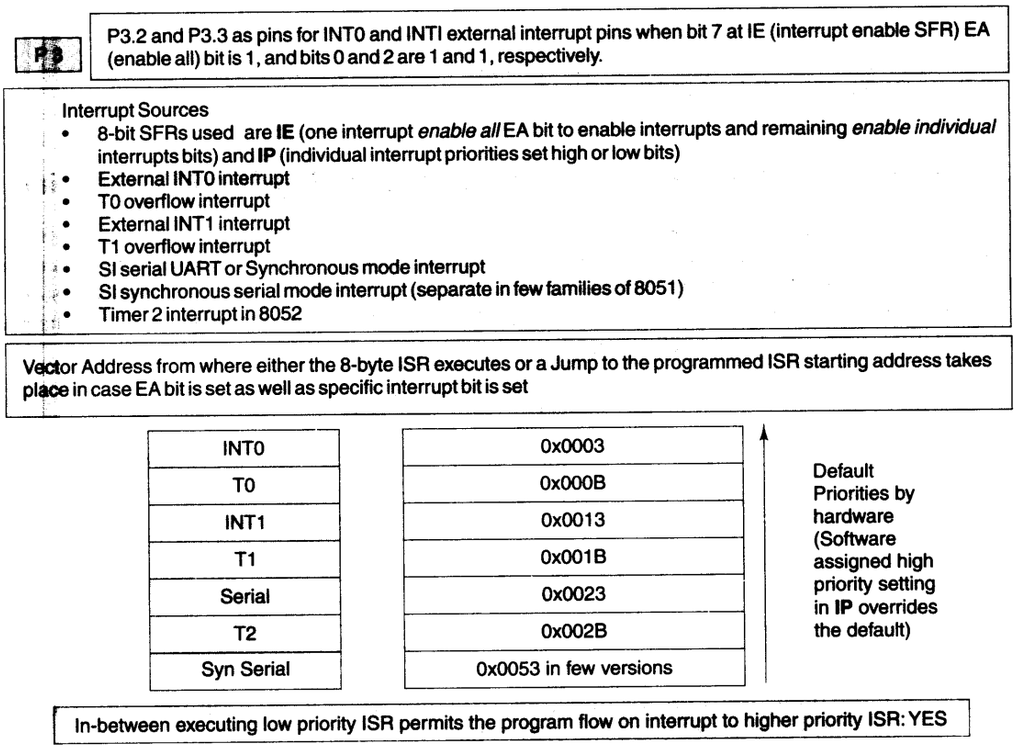


Fig. 2.7 Interrupts in 8051 architecture

SI transmission or receiver interrupts and synchronous mode interrupts occur when SI is programmed using SCON. There are timer T1 and T0 overflow interrupts when T1 and T0 are programmed using TMOD and TCON. There are two external pins for interrupts from peripherals or external circuits.

Two external interrupt pins, INT0 and INT1 at P3.2 and P3.2 can interrupt provided these pins are programmed by the TCON lower 4 bits and the IE register bits IE.2 and IE.0.

8051 hardware sets default priorities for service in the case when multiple interrupts occur concurrently. Priorities by default are in the order INT0, T0 overflow, INT1, T1 overflow, SI (UART mode), T2 (in 8052) and SI (synchronous mode). Using the SFR, called IP (Interrupt Priority) register, at address 0xB8 for the byte and at addresses 0x88 to 0x8C, 0x8D, or 0x8E for the individual bits in the register, an instruction can define that a given interrupt is of higher (=1) or lower priority (=0) among the various interrupts in 8051. [Section 4.6.3]

Using the SFR IE (Interrupt Enable) register at address 0xA8 for bytes and at addresses 0xA8 to 0xAF for the individual bits, a program enables or disables the interrupts (Section 4.4.3).

TCON.3 shows the status of the interrupt at INT1 and auto resets to 0 when the ISR for servicing INT1 interrupt starts. TCON.1 shows the status of the interrupt at INT0 and auto resets to 0 when the ISR for servicing INT0 starts. TCON.2 shows the type of interrupt at INT1 and is 1 if it is of the edge-triggered type, else 0. TCON.0 shows the type of interrupt at INT0 and is 1 if it is edge-triggered type, else 0.

8051 has fixed interrupt vector addresses (Section 4.4.1). An 8-byte address space is provided between two vector addresses. An ISR (Section 4.2) is stored either within these addresses or another ISR is called from these addresses if the ISR is long.

2.2 REAL WORLD INTERFACING

2.2.1 System Bus-based and IO Bus-based IOs for Real World Interfacing

Figure 2.8 shows the interconnections for a simple bus structure when interfacing the processor, memory and IO devices. Three sets of signals – classified as address bus, data bus and control bus defines the system bus. The characteristics of the processor's internal bus(es) differ from that of the system's external bus(es). A system–bus interfacing–design is created according to the needs of the processor signal's timing diagram, speed and the word length for instructions and data.

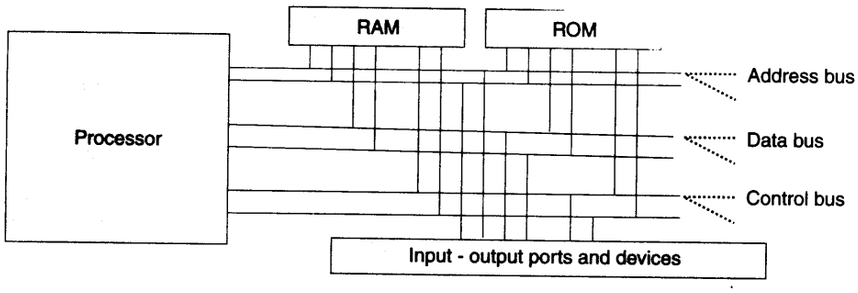


Fig. 2.8 Interconnections for a simple bus structure when interfacing the processor, memory and IO devices using system bus

Address Bus The processor issues the address of the instruction byte or word to memory system through the address bus. The processor execution unit, when required, issues the address of data (byte or word) to memory system through address bus. An address bus of 32 bits fetches instruction or data from an address specified by a 32-bit number.

Example 2.3

1. Let a processor at the start reset the program counter at address 0. Then the processor issues address 0 on the bus and the instruction at address 0 is fetched from memory.
2. Let a processor instruction be such that it needs to load register r1 from the memory address M. The processor issues address M on the address bus and data at address M is fetched.

Data Bus When the processor issues the address of the instruction, it gets back the instruction through the data bus. When it issues the address of the data, it loads the data through the data bus. When it issues the address of the data, it stores the data in the memory through data bus. A data bus of 32 bits fetches, loads, or stores the instruction or data of 32 bits.

Example 2.4

1. When the processor issues address m for an instruction, it fetches the instruction through data bus from address m. [For a 32-bit instruction, word at data bus is from addresses m, m + 1, m + 2 and m + 3.]
2. When an instruction is given to store register r1 to the memory address M, the processor issues address M on the bus and sends the data at address M through the data bus. [For 32-bit data, word at data bus is to the memory addresses M, M + 1, M + 2, and M + 3.]

Control Bus A control bus issues signals to control the timing of various actions during interconnection. These signals synchronize the subsystems. There may be the following:

address latch enable [(ALE) Address Strobe (AS) or address valid (ADV)], memory read (RD) or write (WR) or IO [read (IORD) or write, (IOWR) or data valid (DAV), interrupt acknowledge (INTA) on a request for drawing the processor's attention to an event, or hold acknowledge (HLDA) on an external hold request for permitting use of the system buses, and other control signals as per processor design. Input control signals may be INTR (Interrupt) when external device interrupts the system and HOLD when external device sends a hold request for direct memory access (DMA).

Example 2.5

1. When the processor issues the address, it also issues a memory-read control signal and waits for the data or instruction. A memory unit must place the instruction or data during the interval in which memory-read signal is active and not inactivated by the processor.
2. When the processor issues the address on the address bus, and (after allowing sufficient time for the all address bits setup) it places the data on the data bus; it also then issues a memory-write control signal (after allowing sufficient time for the all data bits setup) to store the bits at memory. The memory unit must write (store) the data during the interval in which memory–write signal is active and not inactivated by the processor.

The buses may have a time division multiplexed (TDM) address and data bits for memories. The interfacing circuit that demultiplexing the buses uses a control signal. [TDM means that in different time slots, there are different sets (channel) of signals.] The system has address signals in one time slot and data bus signals in another. The control signal is called Address Latch Enable (**ALE**) in 8051. The control signal is Address Strobe (AS) in 68HC11. It is address valid, (ADV) in 80196. An ALE, AS or ADV demultiplexes the address and data buses to the devices.

The buses for program and data memory may be multiplexed. The interfacing circuit for the demultiplexing of the buses uses a control signal. The control signal is PSEN in 8051 for demultiplexing common address bus for program and data memory. When the PSEN activates, it signals to read the program memory. When another control signal RD activates, it signals to read the data memory.

Each chip of the memory or port that connects the processor has a separate chip select input from a decoder. The decoder is a circuit that has appropriate bits of the address bus at the input and generates corresponding CS (chip select) control signals for each device (memory and ports) which are at the distinct set of addresses in the system. Demultiplexer and decoder circuits use higher bits of address bus, PSEN and ALE in 8051.

A circuit called glue-circuit, which includes the decoder for interfacing the system buses between the processor, memory and IO devices. Interconnections are through data and address and control bus signals. Understanding timing diagrams of bus signals is essential for appropriate design of the interfacing circuit and fusing (burning) it in a PLD (programmable logic device), GAL or FPGA. Figures 2.4, and 2.9(a) and (b) show the circuits interfacing the memory and ports in 8051 and 68HC11, respectively. The 8051 microcontroller uses an additional signal, PSEN (Program Store Enable), for program codes read from program memory). [This is because of the use of Harvard architecture (Section 2.4.2) for system memories.]

An interfacing circuit consists of decoders and demultiplexers and is designed according to the available control signals and timing diagrams of bus signals. This circuit connects all the units, processor, memory and the IO device through the system buses. It is a part of the glue circuit used in the system and is in GAL (generic array logic) or FPGA.

Figure 2.8 shows a simple diagram of a typical computer system in which buses provide an interconnecting network between the processor, memory, and IO systems. In real world interconnections, the network is formed by buses in the main subsystems.

The system bus interconnects the subsystems, which interconnects the processor with the memory systems and also connects another set of signals called the IO bus. Figure 2.10 shows the system and IO buses. It is a two-level bus architecture. Using an IO bus allows a computer to interface with a wide range of IO devices, without having to implement a specific interface for each IO device. An IO bus can also support a variable number of devices, allowing users to add devices to a system after it has been hardwired. Devices can be designed to interface with the bus, allowing them to be compatible with any system that uses the same type of IO bus. The IO bus creates an interface abstraction that follows the processor to interface with a wide range of IO devices using a very limited set of interface hardware.

Detailed descriptions of popular IO buses and wireless communication are given in Sections 3.10 to 3.13. PCI and USB bus (Section 3.12.2) interfaces to devices are designed to meet the PCI standard and USB (Section 3.10.3) standard.

All that is required is a device driver (Section 4.2.4) in an each operating system—a program that allows the operating system to control the IO device (Section 8.6.1).

The downside of using an IO bus to interface to IO devices is that all the IO devices in a computer must share the IO bus, and IO buses are slower than dedicated connections between the processor and an IO device because the IO buses are designed for maximum compatibility and flexibility.

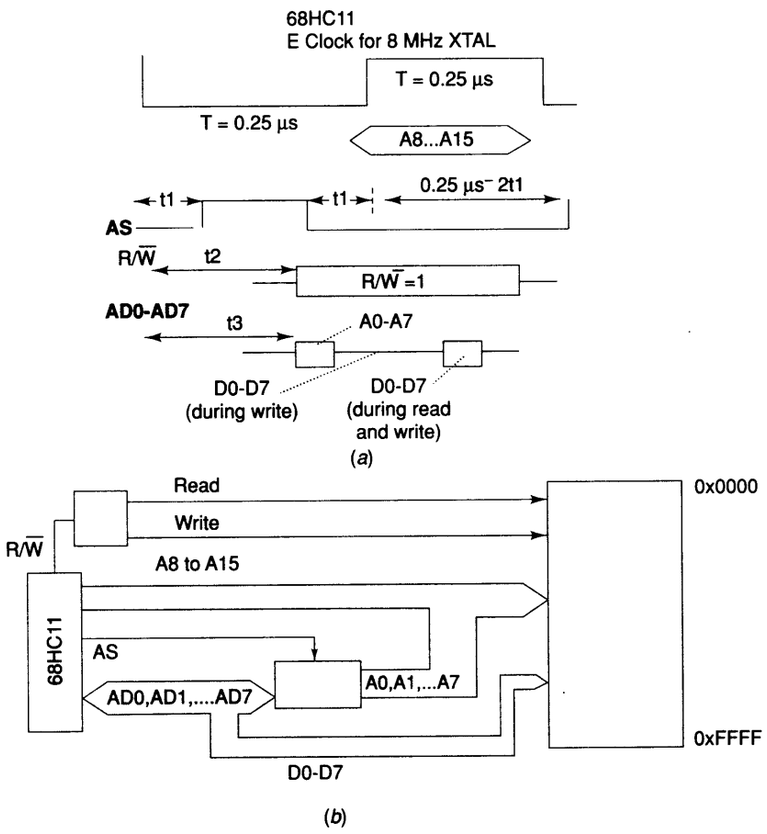


Fig. 2.9 (*a*) Timing of signals from processor when interfacing memory and ports in 68HC11 (*b*) Circuits for the interfacing memory and ports in 68HC11

An interfacing circuit consists of decoders and demultiplexers as well as an IO bus bridge controller. The interfacing circuit is designed as per available control signals and timing diagrams of bus signals. This circuit connects all the units, processor, memory, IO bus bridge controller and the IO devices through the system as well as through the IO buses. IO bus bridge controller may be a part of the glue interfacing circuit used in the system and is in PLD (programmable logic device), GAL (generic array logic) or FPGA.

Multilevel Buses Figure 2.10 shows a two-level bus architecture. Figure 2.11 shows a three-level bus architecture.

2.2.2 IO Addresses of Ports and Devices in Real World Interfacing

Memory Address-Mapped IO Operations Many processors and memory organization require memory-mapped IOs. IO device and port addresses are interfaced such that these are distinct from the memory

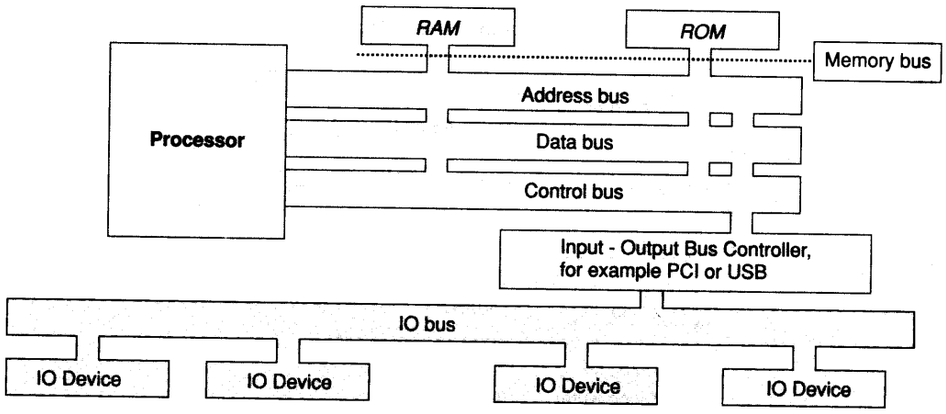


Fig. 2.10 Memory, system bus and IO bus interfacing in a two-level bus structure

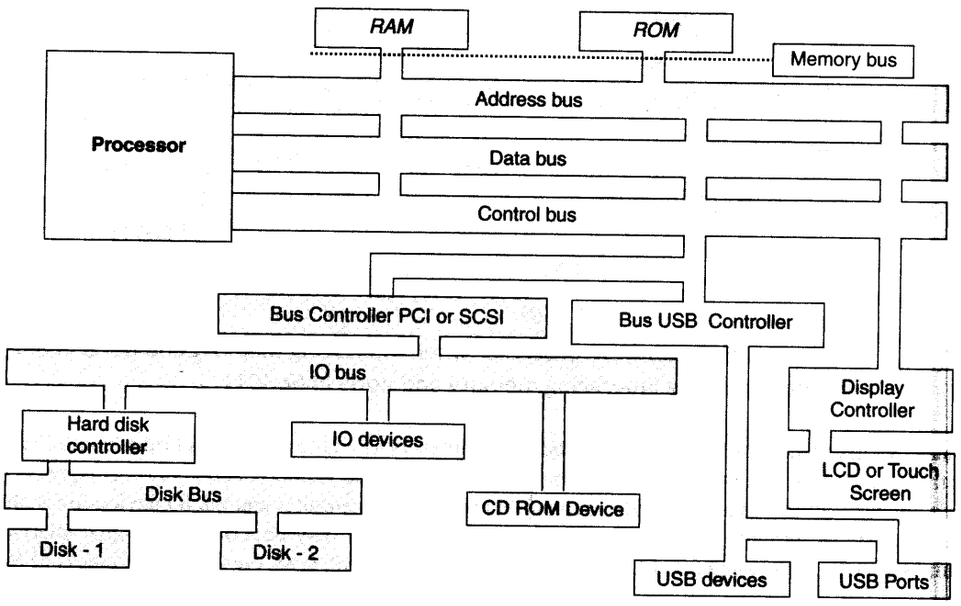


Fig. 2.11 Separate memory and I/O buses to communicate with the memory system, and the I/O system using a bus controller and a separate disk IO bus

addresses. Memory addresses are for data and software, and IO addresses are for the IOs. The following are features of memory-mapped IOs:

- The processor has no separate IO address space for ports and devices.
- The instructions as well as control signals for the operations on bytes at the memory, IO port and device addresses are the same.

- The processor has no separate input–output and memory load–store instructions.
- The arithmetic, logical and bit manipulation instructions that are available for data in memory are also available for IO operations. The processor can directly manipulate the data taken from or stored at the IO port or device. The manipulation of all instructions in the memory can be done using an accumulator, any register or any other memory address where the IO port byte is transferred after, during or before the arithmetic or logical operation.

Almost all microcontrollers, therefore, have no separate instructions for IO processing. The 8051 microcontroller (Section 2.1) is an example of a memory-mapped IO-based processor and memory organization. The 8051, 80196 and 80196 microcontrollers have preassigned device IO addresses for their internal devices and these addresses are not configurable.

Figure 2.12(a) shows that device addresses are within the RAM and are distinct from memory addresses. Motorola processors have no separate instructions for IO processing. Consider another system with a 68HC11 microcontroller.

A configuration is shown in figure. Port A, IO control register PIOC, Port C, B and port control (CTL) registers have addresses between 0x0000 to 0x0004. On-chip RAM is configured between 0x003F to 0x0040. [The port addresses and on-chip RAM are configurable by the bits of the configuration register in 68HC11. For example, the above device addresses can also be re-configured and assigned between 0x0100 and 0x1040.]

IO Addresses Mapped IO Operations Some processor and memory-organization requires IO address-mapped IOs operations. Consider a system with an 80x86 processor. Figure 2.12(b) shows the memory addresses on the left side. It shows the port addresses allocated in IBM PC for timer, keyboard, real time clock and serial port (called COM2) on the right side. This figure shows that device addresses need not be distinct, they can be the same as the memory addresses as a control signal will distinguish between them. The following are features of IO address-mapped IOs:

1. The processor has a separate IO address space for ports and devices.
2. The instructions and control signals for operations on bytes at the memory and IO ports and devices are distinct, making the design simple. IO devices and port addresses are interfaced independently of memory, without considering the memory addresses that are assigned for software and data.
3. The processor has separate input–output (for read and write) instructions and memory load–store (for read and write) instructions.
4. All the arithmetic, logical and bit instructions that are available in memory are first operated using the accumulator and then from there bytes are transferred after an arithmetic or logical operation.

The IO subsystem has input units and output units, also called IO devices. All IO ports and devices have addresses. These are assigned to devices according to the system processor and internal hardware configuration. Direct ALU operations on port byte(s) is not provided.

The addresses of device depend on the system hardware configuration. Most processors follow memory-mapped IOs and process the memory and other devices data with the same instructions. Some processors use IO-mapped IOs; for example, 80x86 processors process these with a different set of instructions (input–output instructions) and control signals.

2.2.3 Device Addresses in Real World Interfacing

During processor instruction, a device when addressed, it gets selected and communicates with system bus or IO bus using a set of addresses. These addresses are selected either as per decoder circuit design or as per the device-driver program for a controller for IO bus. The device addresses access during a read or write operation.

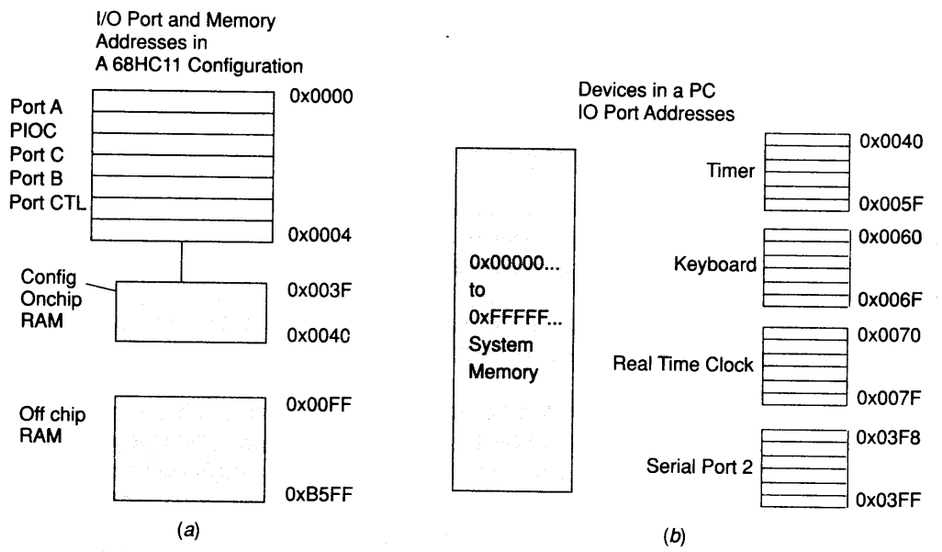


Fig. 2.12 (*a*) Processor and memory organization with I/O devices memory assignments in 68HC11 (with memory mapped IO architecture (*b*) Device Addresses in the 80x86-based IBM PC

1. Device Data Register(s) or RAM buffer(s).
2. Device Control Register(s) to save control and configuration bits.
3. Device Status Register(s) for flag bits as per the device status. A flag may indicate the need for servicing or show an occurrence of device-interrupt.

Each device, and thus each device register, must be allocated addresses at the memory map.

A very important point to remember is that in most cases, each set of IO device addresses is often fixed by the system hardware. A locator or loader cannot reallocate these to any other set of addresses. Also, depending on the device, at a device address there can be one or several device-registers.

A physical or virtual device can be configured to attach or detach from receiving input and sending output. A device address can also be just like a file, record address and can be read only or write only or read and write both.

Example 2.6 gives the details of addresses of the registers of an IO device, called *serial line* or UART device.

Example 2.6

A *serial line device* has the addresses assigned for the device registers. The addresses are fixed by hardware configuration of UART port interface circuit in a system employing an 80x86 processor. They are from 0x2F8 to 0x2FE at COM1 in IBM PC.

1. **(A)** *Two I/O data buffer registers (one for receiving and other for transmitting) are at a common address*, 0x2F8, provided a control bit at address 0x2FBH is 0, **(i)** during read from the address, the processor accesses from the RBR (Receiver Data Buffer Register) and **(ii)** during write to the address, it accesses from the TRH (Transmitter Holding Register) at 0x2F8H. **(B)** Provided a control bit at address 0x2FB is 1, the data of two bytes of *Divisor Latch* are at the addresses, 0x2F8 (LSB) and 0x2F9 (MSB). The divisor latch holds a 16-bit value for

dividing the system clock. This then selects the rate of serial transmission of bits at the line. [A bit in another register (control register) changes 0x2F8 assignment from IO register to lower byte at divisor latch register and 0x2F9 to higher byte.]

2. *Three control registers of device are assigned three addresses* 0x2FA, 0x2FB and 0x2FC for IER, LCR and MCR. **(i)** IER (Interrupt Enabling Register) enables device interrupts. **(ii)** LCR (Line Control Register) defines how and how many bits will be on the line. **(iii)** MCR (Modem Control Register) defines how the modem does a handshake for communication.
3. *Three status registers of the device are also at three addresses* 0x2FA, 0x2FD and 0x2FE as follows: **(i)** IIR (Interrupt Identification Register) at 0x2FA. It has flags that set on a device-interrupt and reset at the system reset and at the servicing of the corresponding device-interrupt. **(ii)** LCR (Line Control Register) at 0x2FD. It defines how and how many bits will be on the line. **(iii)** MCR (Modem Control Register) at 0x2FE. It defines how a modem does a handshake and communicates.

Each IO device is at a distinct address or set of addresses. Each device has three sets of registers; data buffer register(s), control register(s) and status register(s). There can also be one or more device registers at a device address.

2.2.4 Interrupts and IOs

An IO device functioning is slow compared to that of processor. Therefore, an interrupt-driven IO is used. *Interrupts* are the mechanism used by most processors to handle *asynchronous* events.

Essentially, the interrupts allow devices to request that the processor stops what it is currently doing and execute software (called interrupt service routine) to process the device's request, much like a procedure call, ISR initiates by an event at external device rather than by a program instruction.

Interrupts are also used when a processor needs to perform an operation on some IO device and also needs to do other work while waiting for the operation to complete.

Example 2.7

1. Consider a keyboard example. It takes about 10 ms to send the code for a pressed key and thus a maximum of 10 keys can be pressed in 1 s. When does a key input event occurs is not fixed. Intervals, between two events of successive key inputs are also not fixed. In IO mode called interrupt driven mode, when a key is pressed, an interrupt signal RxRDY (receiver data ready) to a processing unit causes the execution of a service routine and the service routine program reads the byte for that code. Figure 2.13(a) shows the method of input from and to a port using RxRDY interrupts.
2. Consider a printer. Assume that a maximum of 300 characters can be printed in 1 s and it thus takes about 0.3 ms to print the code sent at the output by a port. When a print operation completes for a character that is not fixed. Intervals between two events of successive print of the characters are also not fixed. In interrupt-driven mode, when a print action completes, an interrupt signal TxDE (transmission data empty) to the printer processing-unit (print controller) will cause the execution of a service routine and the service routine will then send another byte as output. Figure 2.13(b) shows the method of output from and to a port using the TxDE interrupts.

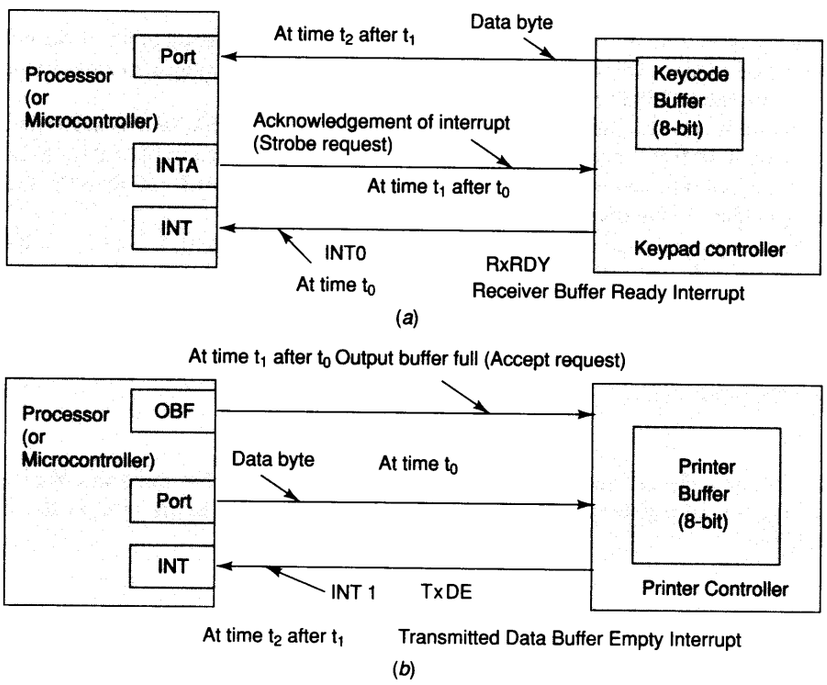


Fig. 2.13 (*a*) Method of Input from and to a port using the RxRDY interrupts
(*b*) Method of Output from and to a port using the TxDE interrupts

2.2.5 Bus Arbitration

There can be several processors as well as several single purpose processors (Section 1.2.1), which share a bus. A single purpose processor can also be a controller. The controller can be part of a device or peripheral.

Figure 2.14(a) shows how the system buses are shared between the controllers, IO processor and multiple controllers for access. Only one of them is granted the bus master status at an instance. In general, there can be a number of DMA or other controllers or processors trying to get access to a bus at the same time, but access can be given to only one of these. Therefore only one processor or controller can be the bus master. A controller is called *bus master* when it has access to a bus at an instance. Any controller or processor can be the bus master at the different instances.

Bus arbitration process refers to a process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus-requesting processor unit. There are three methods, one of which is used in the bus arbitration process. They are *Daisy Chain, Independent Bus Requests and Grant*, and *Polling* methods.

Daisy Chain Method Figure 2.14(b) shows a bus arbitration method called daisy chaining method. It is a centralized bus arbitration process. Bus control passes from one bus master to the next, then to the next and so on. Bus control passes from controller units U0 to U1, then to U2, then U3, and so on. Signals in the arbitration process are as follows: A bus-grant signal (BG) functions like a token, which is first sent to U0. If U0 does not need the signal the bus, U0 transfer it to U1. A controller needing the bus raises a bus-request

(BR) signal. A bus-busy (BUSY) is sent when that controller becomes bus master. When bus master no longer needs the bus, it deactivates BR and BUSY. Another BG is issued and passed from U0 to the controllers one by one lined up according to their priorities.

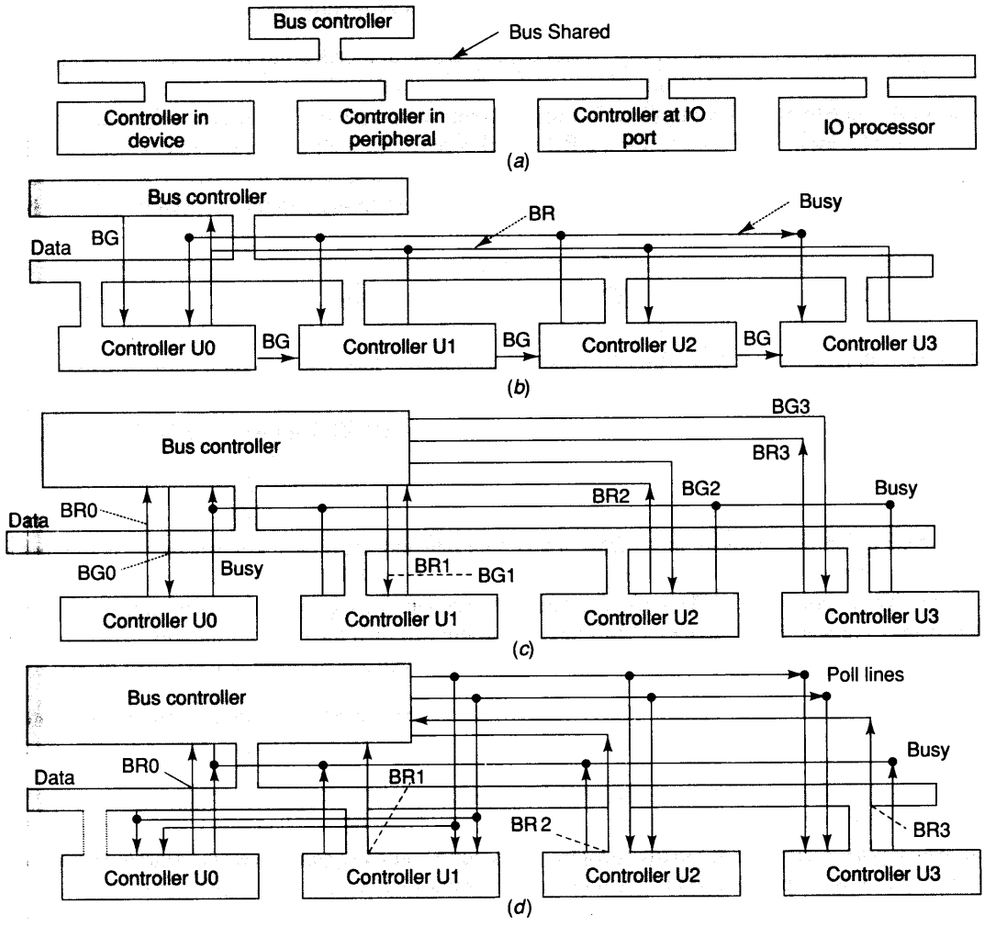


Fig. 2.14 (*a*) System buses shared between the controllers, when IO processor and multiple controllers access the bus, and only one of them granted bus master status at any one instance (*b*) Bus Arbitration by daisy chaining method (*c*) Bus Arbitration by independent bus request method (*d*) Bus Arbitration by polling bus method

The advantage in this is that at each instance of bus access, the i^{th} controller gets highest priority compared to $(i + 1)^{th}$. The controllers and processor priorities for granting bus access (bus master status) are fixed.

Independent Bus Request Method Figure 2.14(c) shows the bus arbitration called independent bus request method, in which each controller has separate BR signals, BR0, BR1, ..., BRn. Also, there are separate

BG signals, BG0, BG1, ..., BGn for each controller. An i^{th} controller sends BRi (i^{th} bus request signal) and when it receives BGi (i^{th} bus grant signal), it uses the bus and activates BUSY signal. Any controller, which finds an active BUSY, does not send a BR from it. The advantage here is that the i^{th} controller can be programmed to give the highest priority to the bus and the priority of a controller can be programmed dynamically.

Bus Polling Method Figure 2.14(d) shows the bus arbitration called bus polling method with two poll lines for four controllers. A poll count value is sent to the controllers and incremented to provide bus access to the next. Assume there are 8 controllers. Three poll count signals p2, p1, p0, successively change from 000, 001, ..., 110, 111, 000, ... If on count = i, a BR signal is received, then counts increment stops, and BUSY activates when that controller becomes the bus master. When BR deactivates then BUSY also deactivate and count increment starts. The advantage is that the controller next to the current bus master gets the highest priority to access the bus after the current bus master finishes its operations.

2.2.6 Interfacing Examples with Keyboard, Displays, D/A and A/D Conversions

Keyboard Figure 2.15(a) shows an interface to a keyboard. Two signals from a keyboard controller are KBINT and TxD. KBINT is interrupt due to RxRDY signal from keyboard controller. TxD is the serial UART data output of a controller connected to RxD at SI in 8051, or UART Intel 8250, or UART 16550, which includes a 16-byte buffer.

Bounces create on pressing a key. This is due to a natural spring-like action. Each bounce results in a false pulse. The keyboard controller has a hardware debouncer to neutralize the false pulse. The keyboard controller has a counter, which continuously increments at a certain rate and scans each key whether it is in pressed or released state. It has an encoder to encode the keyboard output for a ROM. The ROM then generates an ASCII code output for the pressed key. The code also takes into account the meaning of multiple keys when they are simultaneously pressed. For example, if shift key is also pressed then the code for an upper case character is generated. The code bits are serially transferred to TxD output, which is received at RxD input of SI.

Display Section 1.3.8 described the LCD, LED and touchscreen displays. Figure 2.15(b) shows an interface circuit to an LCD display controller. Section 3.3.4 gives details. There are 8 output data and 3 bits for E, RS and R/W. One 8-bit port is used for output data. Another port is used for 3 bits.

Digital Analog Converter Section 1.3.7 described the DAC (also called D/A). A D/A needs a PWM circuit, which is an internal device in microcontroller. A pulse width register (PWR) is programmed according to a required analog output. A counter/timer device generates two internal interrupts: one on timer overflow and another after an interval proportionally equal to PWR. On the first interrupt the output becomes 1 and on the second it becomes 0. An external integrator generates the analog output as per the period of output 1 (period between the first and second interrupts) compared to the total period of output pulses (period between successive first interrupts). Figure 2.15(c) shows an interface to an external D/A. The external D/A is used as an alternative when PWM is not used.

Analog to Digital Converter Section 1.3.7 described an ADC (also called A/D). An n-bit A/D needs (i) a start pulse for converting using a short duration single pulse generator circuit, (ii) a sample hold amplifier circuit to hold the signal constant during the conversion period and (iii) positive and negative voltage references for providing the reference potential difference for conversion of analog input into *n*-bits. A four or eight channel A/D is inbuilt in microcontrollers. An external (ADC), for example, ADC0808, can also be used with interfacing similar to that of the ports. Figure 2.15(d) shows an interface to an external A/D when internal A/D not used.

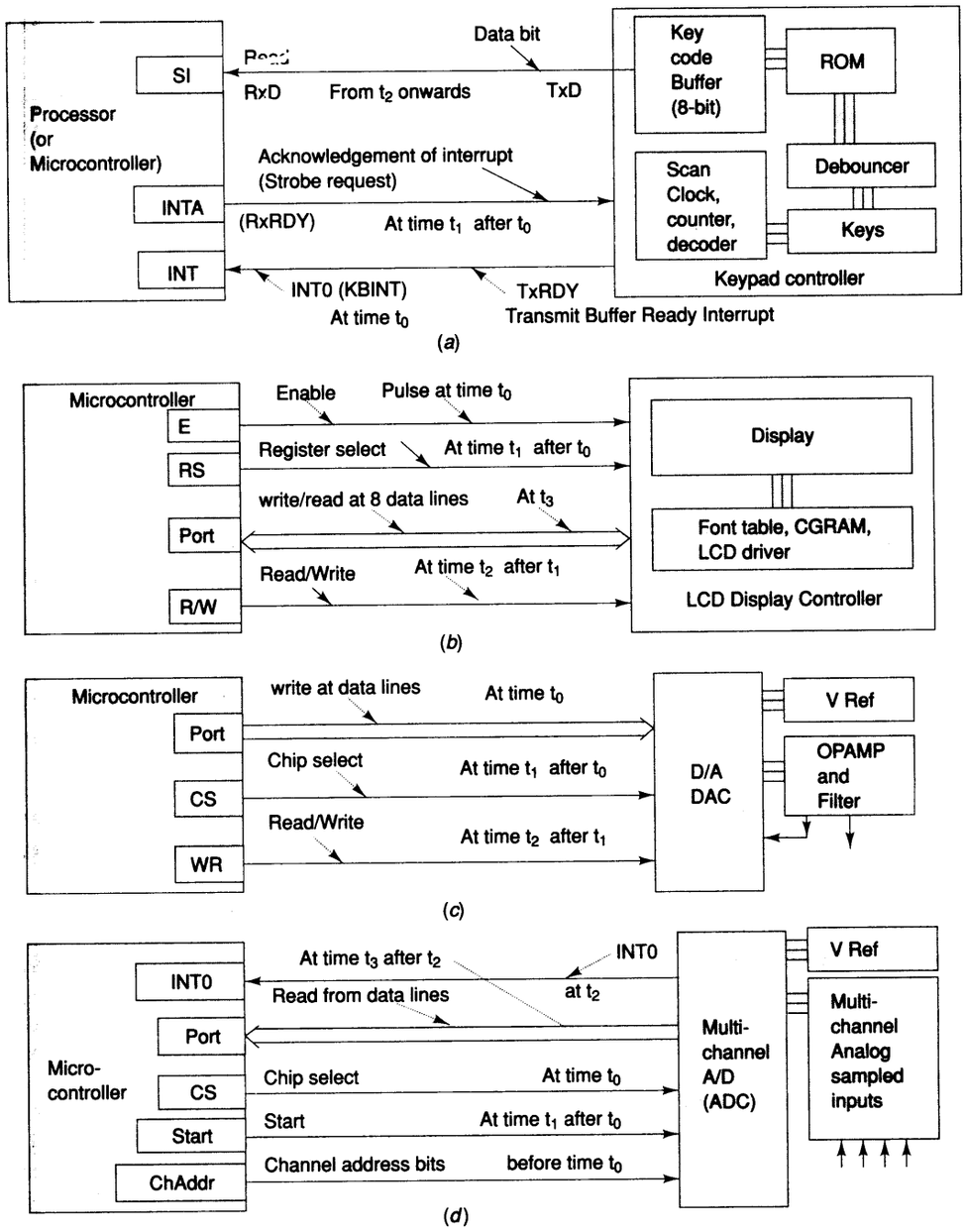


Fig. 2.15 (*a*) Interfacing to a keyboard using keyboard controller (*b*) Interfacing to a LCD display controller (*c*) Interfacing to D/A (DAC) when internal PWM not used (*d*) Interfacing to external A/D (ADC) when internal ADC not used

2.3 INTRODUCTION TO ADVANCED ARCHITECTURES

Figure 2.16 shows an organization of various processor units. The units, which are shown with the dashed boundary are present in high performance processors. External address, data and control buses interface with the processor and connect to external memory units, ports and devices.

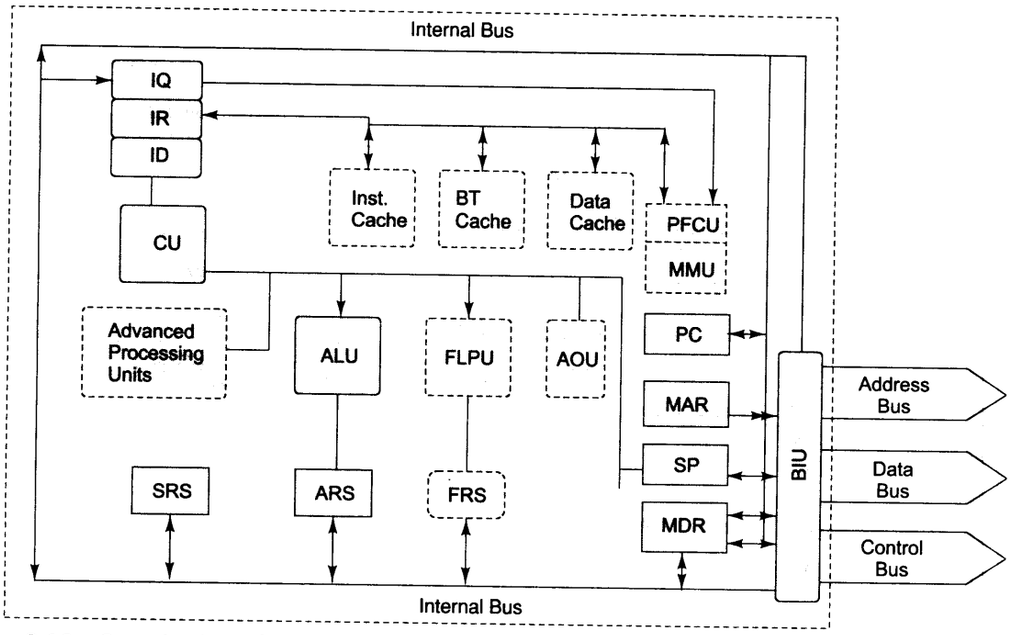


Fig. 2.16 Organization of various processor units. The units, which are shown with the dashed boundary are present in processors having high-performance advanced architecture

The following are the general features present in most processors.

1. *Fixed Instruction Cycle Time:* This is the time taken by a processor to execute a simple instruction, which is ~1 μs for the 8051 processor operating at ~12 MHz, and 0.9 μs per MHz clock rate for the ARM9 processor. A system designer uses the instruction cycle time as an indicator to select processor clock speed as per the application. For example in applications that need fast processing, the ARM9 processor at 100 MHz would be considered suitable; for other applications for which slower processing will suffice the 8051, 68HC11 or 80196 can be chosen.
2. *Internal bus width:* An ALU gets inputs through the internal buses. Bits in a single operand to ALU (during a single arithmetic or logical operation) are equal to the bus width. A 32-bit bus will facilitate the availability of arithmetic operations on 32-bit operands in a single cycle. The 32-bit bus becomes a necessity for signal processing and control system instructions. When the bus-width is 32 bits, it reads or writes an integer of 32 bits and will process about four times faster than when the width is 8. An internal bus of 128 bits is present in SHARC and 64 bits in Pentium.
3. *Program-counter* (PC) *bits and reset value*: The number of PC bits decides the maximum possible size of the physical memory that can be accessed by the processor. The reset value tells the designer

the initial program address from where the program runs on a system reset or power up. The processor will start execution from that address. [The initial instruction pointer and code segment register bits decide the initial program's memory address in 80x86 processors.]

4. *Stack-pointer bits and initial reset value*: Stack pointer values must point to addresses of the words stored at the *stack*. These addresses must be within the ones allocated for stack in the memory. The software designer defines an initial reset value and sets the beginning stack pointer accordingly.

Table 2.1 lists the structural units in a general-purpose processor. It lists the functions of each.

Table 2.1 General structural units in a processor architecture

Structural	*Unit*	*Functions*
MAR	Memory address register	It holds the address of the byte or word to be fetched from external memories. Processor issues the address of instruction or data to MAR before it initiates fetch cycle.
MDR	Memory data register	It holds a byte or word fetched (or to be sent) from (to) an external memory or IO address.
System buses	Internal Bus	It internally connects all the structural units inside the processor. Its width can be 8, 18, 32, 48 or 64 bits.
	Address bus	An external bus that carries the address from MAR to memory as well as to IO devices and other units of system.
	Data bus	An external bus that carries, during a read or write operation, the bytes for instruction or data from or to an address. The address is determined by MAR.
	Control bus	An external set of signals to carry control signals to processor or memory or device.
BIU	bus interface unit	An interface unit between processor's internal units and external buses.
IR	Instruction register	It sequentially takes instruction codes (opcode) to execution unit of processor.
ID	Instruction decoder	It decodes the instruction received at the IR and passes it to processor CU.
CU	Control unit	It controls all the bus activities and unit functions needed for processing.
ARS	Application register set	(a) A set of on-chip registers used during processing of instructions of an *application* program or (b) a register window, (c) a subset of registers with each subset storing static variables of a software-routine or (d) a register file associated to a unit such as ALU or FLPU.
ALU	Arithmetic logical unit	A unit to execute arithmetic or logical instructions according to the current instruction present at IR.
PC	Program counter	It generates an instruction cycle by sending the address defined by it to memory through MAR. It auto-increments as the instructions are fetched regularly and sequentially. It is called instruction pointer in 80x86 processors.
SP	Stack pointer	A pointer for an address, which corresponds to a stack-top in memory.

2.3.1 Architecture of the Advanced Processors

Figure 2.16 shows the additional units in boxes with dashed boundary and these units are present in advanced processor architectures (high performance processors). Table 2.2 lists the advanced architecture structural units in a processor organization of general-purpose processor. It lists functions of each unit.

Table 2.2 Structural units in an advanced processor architecture

Structural	*Unit*	*Functions*
Instruction level parallelism units	ILP	For instruction level parallelism (Section 2.5), the multistage pipeline processing, multiline superscalar processing, and dual, quad or multicore processing speeds up the performance from one instruction per clock cycle[1].
IQ	Instruction queue	A queue of instructions so that the IR does not have to wait for the next instruction after one has been processed.
PFCU	Prefetch control unit	A unit that controls the *fetching* of data into the I- and D-caches in advance from the memory units. The instructions and data are delivered when needed by the processor's execution unit(s). The processor does not have to fetch data just before executing the instruction. Pre-fetching unit improves performance by fetching instructions and data in advance for processing. Caches along with a MMU improve performance by giving the instructions and data fast to the processor execution unit.
I-Cache	Instruction cache	It sequentially stores, like an instruction queue, the instructions in FIFO mode. It lets the processor execute instructions at great speed using PFCU compare to external system-memories, which are accessed at relatively much slower speeds.
BT Cache	Branch target cache	It facilitates ready availability of the next instruction-set when a branch instruction like *jump, loop, or call* is encountered. Its fetch unit foresees a branching instruction at the I-cache.
D-Cache	Data cache	It stores the prefetched data from external memory. A data cache generally holds both the key (address) and value (word) together at a location. It also stores write-through data when so configured. Write-through data means data from the execution unit that transfer through the cache to external memory addresses.
MMU	Memory-management Unit	It manages the memories[2] such that the instructions and data are readily available for processing.
SRS	System register set	It is a set of registers used while processing the instructions of the supervisory system program.
FLPU	Floating point processing unit	A unit separate from ALU for floating point processing, which is essential in processing mathematical functions fast in a microprocessor or DSP.
FRS	Floating point register set	A register set dedicated for storing floating point numbers in a standard format and used by FLPU for its data and stack.
MAC	Multiply and accumulate unit	There is also a MAC[3] units for multiplying coefficients of a series and accumulating these during computations.

(*Contd*)

Structural	*Unit*	*Functions*
AOU	Atomic operation unit	It lets a user (compiler) instruction, when broken into a number of processor instructions called atomic operations, finish before an interrupt of a process occurs. This prevents problems from arising out of shared data between various routines and tasks.

1. Instruction cycle time becomes several times less than the processor clock cycle time.
2. The MMU manages the pages in the RAM memory as well as the copies in internal and external caches. Managing has to be done in such a way that when the instructions execute, there are minimum number of page and cache faults (misses).
3. MAC units are invariably needed in DSPs. [Section 2.3.5]

Advanced processor circuits consist of RISC architecture. It improves performance by executing most instructions in a single clock cycle (by hardwired implementation of instructions), by using multiple register-sets, windows and files and by greatly reducing dependency on the external memory accesses for data due to the reduced number of addressing modes for arithmetic and logic instructions. An RISC has only a few addressing modes for arithmetic and logic instructions. It does not have the following addressing modes: indirect (index), auto-index, and index-relative for ALU instructions. It does not have a second operand fetched by the immediate addressing mode for arithmetical and logical instructions.

Advanced processor circuits consist of a floating-point unit; FRSs process mathematical functions faster and with greater precession than when employing an integer-processing ALU.

Advanced processing units include the instruction pipelining unit, which improves performance by processing instructions in multiple stages. Pipelining allows a processor to overlap the execution of several instructions so that more instructions can be executed in the same period of time. Section 2.5.1 will describe multiple stages of instruction execution and will describe how instruction level parallelism (ILP) further improves processor performance.

Figure 2.17 shows how instructions flow through the pipeline.

In cycle 1, the first instruction I_1 enters the instruction fetch (IF) stage of the pipeline and stops at the pipeline latch (buffer) between the instruction fetch and instruction decode (ID) stage. In cycle 2, the second instruction I_2 enters the instruction fetch stage, and I_1 proceeds to the instruction decode stage. In the cycle 3, I_1 enters the register (inputs) read (RR) stage, instruction I_2 is in the instruction decode stage, and instruction I_3 enters the instruction fetch stage. In fourth cycle, I_1 moves to execute stage and in fifth cycle to result write back stage.

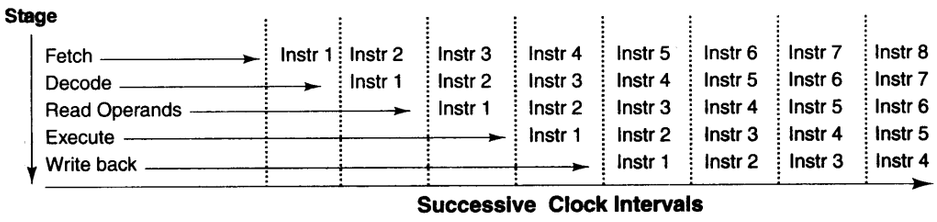


Fig. 2.17 Instruction flow in a pipeline of an advance architecture processor

Instructions proceed through the pipeline at one stage per cycle until they reach the register (result) write-back (WB) stage, at which point execution of the instruction I_1 (instr1 in figure) is complete. Thus, in cycle 6

in the example, instructions I_2 through I_6 are in the pipeline, while instruction I_1 has completed and is no longer in the pipeline. A 5-stage pipelined processor is still executing instructions at a rate (throughput) of one instruction per cycle, but the latency of each instruction is now 5 cycles instead of 1. The faster execution takes place as cycle time now can be one-fifth or less than unpipelined case.

2.3.2 80x86 Architecture

The first four generations of 80x86 are 8086, 80286, 80386 and 80486. The first processor in the 80x86 family of processors is the 16-bit 8086 (1981). The 80x86 has a 32-bit architecture since 80386. Pentium is the fifth generation architecture (1994) based on the 32-bit 80386. Pentium 4 is of seventh generation and Xeon and Core2 are eighth generation architectures. Core2 means dual core architecture. The 80x86 architecture processors have become popular since their application in the IBM PC (personal computer). Itanium is based on 64-bit architecture, which simulates the 80x86 architecture.

The features of 80x86 architecture are as follows:

1. The original 8086 architecture consists of general purpose registers AX, BX, CX and DX. Each can be considred as two 8-bit registers. For example AX as AL (A lower byte) and AH (A upper byte). A 32-bit extension has EAX, EBX, ECX and EDX. EAX registers. Each can be considred as two 16-bit registers. AX then has a lower 16-bit EAX. Figure 2.18 shows the 80x86 architecture registers.
2. The 8086 architecture provides for code, data, stack and stack segmentations. The original 8086 architecture consists of four segment registers CS, DS, SS and ES to enable access to memory assigned to different segments.
3. IP is an instruction pointer, of a 16-bit address, and CS contains a 16-bit program code segment address for 16 upper bits of address.
4. SI contains index of source operand and DI contains 16-bit index of destination. BP is memory offset pointer of 16 bits address and DS contains a 16-bit data memory segment address upper bits.
5. 16-bit or 32-bit or 64-bit words store as little endian. Data need not be aligned at the addresses in multiples of 2 or 4 and can start from any address.

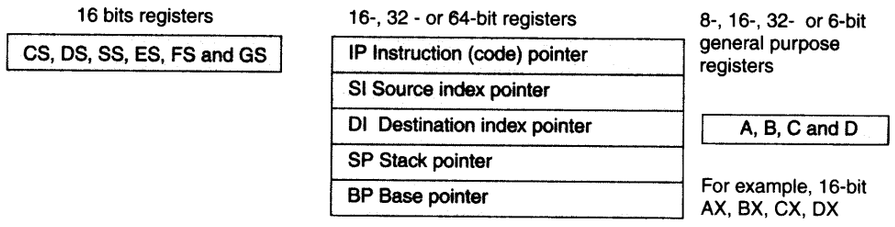


Fig. 2.18 80x86 architecture registers

6. The 80x86 mainly uses two address arithmetic and logic instructions. This means that the accumulator is not the only register to accumulate ALU result, which in turn means that a register operand (AX or BX or CX or DX) can be a destination as well as the first source operand.
7. A memory address can be the first or second operand, a characteristic of CISC addressing modes for ALU instructions.
8. The present generation 80x86 architecture decodes a CISC instruction and creats microoperations that implement on a microarchitecture of RISC.
9. The small number of general registers (also inherited from 8085) has made register-relative addressing (using small immediate offsets) an important method of accessing operands, especially on the stack.

10. The 80x86 has IO mapped IO. An IO address is of 16 bits for an IO byte. Processors of the Intel 8086 family process and access IO units and IO devices by the separate IN and OUT instructions. The IO mapped IO processors have a separate set of addresses for accessing inputs and outputs. It simplifies the IO units interfacing circuit that connects to the processor.
11. The 8086 supports 256 interrupt levels for the hardware as well as software and supports nested interrupts. This means that an ISR can be interrupted and a higher priority ISR can execute in between.
12. The new generation 8086 architecture supports a mode called real mode. Real mode supports direct access without segmentation to peripheral devices and basic input output subroutines (BIOS). Real mode supports 20-bit segmentation instead of 16-bit. The segment register has only the upper 16 bits. The lower bits are 0s.
13. A mode called 32-bit protected mode is also provided and supports pages in memory.
14. 8086 supports many OSs, including Windows and multitasking operating systems.
15. The latest 80x86 architectures support thread handling, integer SIMD and SIMD extension instruction sets.

Program routines and processes can have different segments. For example, a program code can be segmented and each segment stored at a different memory block. A *pointer* address points to the start of the memory block storing a segment and an *offset* value is used to retrieve a memory address within that segment. The data can also be segmented with each segment at different blocks. Similarly, strings can be segmented.

The 80x86 architecture is a widely used architecture. The data are not aligned and save as little endian. It has general purpose pointers and segment registers and supports memory segmentation and paging. There can be different segments at the memory for different functions and processes (tasks). These can comprise different segments for data and different segments for the stacks.

2.3.3 ARM

Detailed information on ARM is at http://www.arm.com. A brief description of ARM architecture and features that makes it important for embedded systems, such as digital and video cameras and mobile phones, is give here.

Figure 2.19 shows ARM7 registers and a three stage pipeline architecture. ARM has registers R0 to R15. R15 also functions as a program counter. R14 functions as a link register. It has CPSR (current program status register) and SPSR (saved program status register).

The main features of ARM are as follows:

1. It has 32-bit architecture but also supports 16-bit or 8-bit data types. It supports 16-bit instructions also in Thumb® mode. It supports Jazelle Java execution accelerator.
2. ARM is programmable as little endian or big endian data.
3. ARM provides the advantage of using a CISC in terms of functionality, along with the advantage of an RISC in terms of faster program implementation as well as reduced code lengths. It implements faster because the register word instantly availability to execution-unit. Code lengths are reduced because most instructions use registers as operands. Few bits in the instruction specify a register as operand. 8, 16 or 24 bits specify a memory address as operand and the displacement bits in the instruction.
4. ARM7 and ARM9 microprocessors have a combination of RISC and CISC features. ARM supports a complex addressing modes-based instruction set. ARM processor has an RISC core for processing. There is an in-built compilation unit. It first compiles the CISC instructions into RISC formats, which are then implemented by the RISC core of the processor. Internally, the implementation for many instructions is like in a RISC (without the micro-programmed unit).
5. ARM7 has Princeton memory architecture; ARM9 has Harvard architecture. [Section 2.4.2]

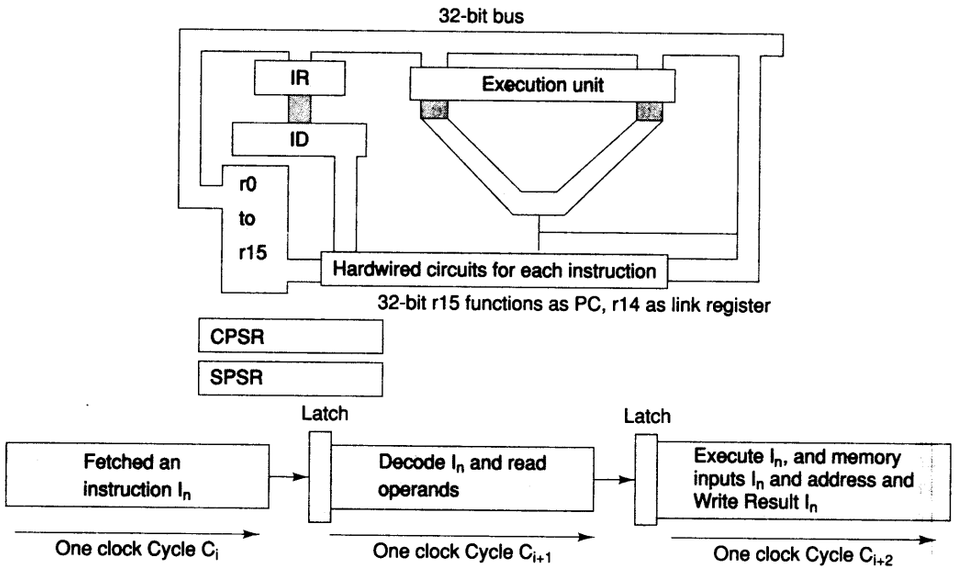


Fig. 2.19 ARM7 registers and three stage pipeline architecture

6. ARM debug and trace tools quickly debug real-time software, and trace instruction execution and associated program data at full core speed.
7. A wide choice of development tools and of simulation models for leading EDA (Electronic Design Automation) environments and excellent debug support for SoC design are available.
8. ARM codes are forward compatible with higher versions. For example, ARM7 codes are forward compatible with ARM9, ARM9E and ARM10 processors as well as with Intel XScale micro-architecture. ARM9E and ARM 10 families use a Vector Floating Point (VFP) ARM coprocessor, which adds full floating point operands. VFP also provides fast development in SoC design when using tools like MatLab®. Applications are in image processing (scaling), 2D and 3D transformations, font generation and digital filters.
9. ARM permits programming by an additional instruction set designed for 16-bit operations. Thumb is an industry standard instruction set, which enables 32-bit performance at the 8/16-bit system cost in terms of memory needs. This provides typical memory savings of up to 35%, over the equivalent 32-bit code, while retaining all the benefits of a 32-bit system (such as access to a full 32-bit address space). There are no overheads (in terms of time and memory) in moving between Thumb and the normal ARM state of the codes. The two states are compatible on a normal basis. This gives the code designer complete control over performance and code-size optimization.
10. ARM uses an Intelligent Energy Manager (IEM) technology. It implements advanced algorithms to optimally balance processor workload and energy consumption. It maximizes system responsiveness. IEM works with the operating system and mobile OS. An application running on a mobile phone dynamically adjusts the required CPU performance level.
11. ARM processors use the AHB (AMBA Advanced High Performance Bus) interface. AMBA is an established open source specification for on-chip interconnects. [Section 3.12.3] AMBA serves as a framework for SoC designs and development of IP cores. It provides a high-performance and fully

synchronous back plane. (Back plane has additional set of controllers, which can access each other through another comon bs, which is distinct from system bus. The multilayer AHB in version ARM926EJ-S and all members of the ARM10 family represent a significant advancement. They reduce accesslatencies and increase the access-andwidth in a multimaster (multiple controllers accessing the bus as master) system.

Instruction Set – ARM7 Processors have the following type of instruction sets. The ARM7 in version with suffice T has instruction set called Thumb® intruction set support.

1. *Data Transfer Instructions* Given below are the instructions for transfer between register-memories. The memory address is as per a register used in index or index-relative or post auto-index addressing mode.
   (a) load in register a word (LDR)
   (b) store from register a word (STR)
   (c) set a memory address in a register (ADR). Addrss is of 12 bits. [Alternative for 16 bits address setting in a register is using any register or r15 in an arithmetic operation.]
   (d) load in register a byte (LDRB)
   (e) store from register a bute (STRB)
   (f) store from register a half word (STRH) [A word in ARM is of 32 bits.]
   (g) load in register a half word as such or signed half word (LDRH or LDRSH).
   The following are the instructions for a word transfer between registers:
   (a) Move (MOV)
   (b) Move reverse (MVR)

A load or move or store instruction can be conditionally implemented. For example, MOVLT r3, #10. The immediate operand 10 will transfer to r3 provided a previous instruction for comparison showed the first source as less than the second. Conditions are LT (signed number less than), GT (signed number greater than), LE (signed number less or equal), EQ (equal), NE (not equal), VS (overflow), VC (no overflow), GE (signed number greater than or equal), HI (unsigned number higher), LS (unsigned number lower), PL (plus, nor Negative), MI (minus), CC (carry bit reset), and CS (carry bit set).

2. *Bit Transfer or Manipulation Instructions*
   (a) Register-bits Logical Left Shift (LSL)
   (b) Register-bits Logical Left Arithmetic Shift (ASL)
   (c) Register-bits Logical Right Shift (LSR)
   (d) Register-bits Logical Right arithmetic Shift (ASR)
   (e) Register-bits Rotate Right (ROR)
   (f) Register-bits Rotate Right with carry also extended for rotating (RRX).
3. *Arithmetical and Logical Instructions* The following are the instructions for arithmetical operations. Each uses three operands from the registers. One source may, however, be immediate operand addressing in addition and subtraction.
   (a) Add without carry two words and put result at the third operand (ADD)
   (b) Add with carry two words and put result at the third operand (ADC)
   (c) Subtract without carry two words and put result at the third operand (SUB) [Carry bit used as borrow.]
   (d) Subtract with carry two words and put the result is at the third operand (SBC)
   (e) Subtract reverse (second source with the first) without carry two words and put result is at the third operand (RSB) [Carry bit used as borrow.]
   (f) Subtract reverse with carry two words and the result is in the third operand (RSC)
   (g) Multiply two different registers and put result is at the destined register (MUL)

(h) Multiply two source registers and add the result with the third source register and accumulate the new result in a destined register (MLA) [There are four operand registers.]

The following are the instructions for logical operations:

(a) Bit wise OR two words and put result at the third operand (ORR)
(b) Bit wise AND two words and put result at the third operand (AND)
(c) Bit wise Exclusive OR two words and put result at the third operand (EOR)
(d) Clear a Bit (BIC). [There is one source for the bits; a second source for the mask and the result is put at the third operand.]

An arithmetical or logical instruction can be conditionally implemented. For example, SUBGE r1, r3, r5. The operand from r3 is subtracted from r5 if the GE condition resulted in earlier operation for test or comparison.

The following are the instructions for *compare and test operations.* The result destines to CPSR, which stores four condition bits, N, V, C, and Z.

(a) Bit-wise Test two words (TST)
(b) Bit-wise negated test between two words (TEQ)
(c) Compare two words and put result at the CPSR condition bits (CMP)
(d) Compare two negative words and put result at the CPSR condition bits (CMN)

4. *Program-Flow Control Instructions* The following are the instructions for branching operations. A branching instruction can be conditionally implemented. Branch to an address relative to PC word in r15 (*B*). '*B* #1A8' means add 0x1A8 in PC and change the program flow. 'BGE #100' means that if a GE condition resulted on a previous compare or test, then add 1A8 in the PC. There are similar instructions for different conditions of the processor status flags (at CPSR). [PC is r15.]

Example 2.8

This example gives an assembly language program example for the ARM.

Consider the problem of additing three numbers, x, y and z (= 127, 29 and 40) and storing the result at a memory address, M for a [a = x + y + z.] Using the instructions of the above instruction-set, the assembly language codes will be as follows.

```
1. BEGIN: MOV r2, #0x007F  ; Transfer 127 into processor register r2.
2.        MOV r3, #0x001D  ; Transfer 29 into processor register r3.
3.        MOV r4, #0x0028  ; Transfer 40 into processor register r4.
4.        MOV r1, #0x000   ; Transfer 0 into processor register r1.
5.        ADD r1, r1, r4   ; Add the register r4 word into the r1.
6.        ADC r1, r1, r3   ; Add the register r3 word along with the carry (if any) from previous
                             addition into the r1.
7.        ADC r1, r1, r2   : Add the register r2 word along with the carry (if any) from
                             previous addition into the r1.
8.        ADR r5, 0x800    ; Set the address into r5. Memory address M set 0x800.
9.        STR [r5], r1     ; Store the r1 at the address pointed by r5.
```

Table 2.3 gives features and comparison of the exemplary high performance ARM family of processors.

1. ARM9™ Thumb® family supports Windows CE, Palm OS, Symbian OS, Linux and other OS/RTOS. There is Palm OS support in ARM920T and ARM922T processors. ARM 940T has a memory Protection Unit (MPU) and a support to a range of Real-Time Operating Systems including VxWorks.

2. ARM7 and ARM9 integrates instruction and data caches.
3. ARM architecture refers specifically to the architectural instruction sets and programmers models, such as ARMv5TE, ARMv5TEJ and ARMv6 architecture in ARM11.
4. ARMv4T (version 4 Thumb) microarchitecture is common to ARM7, ARM9, ARM 10 and ARM 11 families. The term ARM microarchitecture refers specifically to the implementation of architectures such as the ARM9™ family of cores and the ARM10 family of cores. For example, ARM926EJ-S™ core and the ARM1020E™ core are CPU products based on those earlier microarchitectures. An enhancement of v4T architecture is ARMv5TE architecture (introduced in 1999). It has ARM DSP instruction set extensions that improves the speed of instruction set by up to 70% for audio DSP applications. [Certain applications need microcontroller data processing features as well as DSP features in a single processor in place of the multiprocessor system.]

Table 2.3 Comparative features of ARM versions

Feature	*ARM7™ Thumb® Family*	*ARM9™ Thumb® Family*	*ARM11*
Family members Example	(a) ARM7TDMI® (Integer Core) (b) ARM7TDMI-S™, (Synthesizable version of ARM7TDMI) (c) ARM7EJ-S™ (Synthesizable core with DSP and Jazelle technology) and ARM720T™ (cached processor macrocell, 8K Cached Core with Memory Management Unit (MMU) supporting operating systems (OSes) Windows CE, Palm OS, Symbian OS and Linux)	(a) ARM920T (Dual 16 k caches with MMU support, OSes). (b) ARM922T (Dual 8 k caches for applications, support for multiple OSes). (c) ARM940T™ (Dual 4 k caches for embedded control applications running an RTOS)	Families with ARMv6 instruction set architecture that include the Thumb® extensions for code density, Jazelle™ technology for Java™ acceleration, ARM DSP extensions and SIMD media processing extensions. MMU support, OSes and Palm OS
Core with ARM® and Thumb® instruction sets	32-bit RISC core	32-bit RISC processor core super scaling 5-stage integer pipeline. 8-entry write buffers. It avoids blocking the processor on external memory *writes*	32-bit RISC processor core with 8-stage integer pipeline, static and dynamic branch prediction, and separate load-store and arithmetic pipelines to maximize the instruction throughput
Application domain	Cost and power-sensitive consumer applications for example, Personal audio MP3, WMA, AAC players, entry level mobile phone, two way pagers, still digital camera, PDAs	Set-top boxes, home gateways, games consoles, MP3 audio, MPEG4 video videophones, portable communicators, PDAs, next-generation hand-held products, digital consumer products, imaging products, desktop printers, still picture cameras, digital video cameras, automotive telemetric and infotainment systems	Battery-powered and high-density embedded applications. Embedded SoCs of latest generation of wireless and consumer applications. Addresses the requirements of embedded application processors, advanced OSes and multimedia, such as audio and video CODECs. Consumer devices include 2.5G and 3G mobile phone handsets, PDAs and multimedia wireless

(*Contd*)

Feature	*ARM7™ Thumb® Family*	*ARM9™ Thumb® Family*	*ARM11*
			devices, home consumer applications such as imaging and digital camera applications, home gateway and network infrastructure equipment including voice over IP and broadband modem
Performance	130 MIPS using Dhrystone 2.1 benchmark in typical 0.13 μm process	Achieves 1.1 MIPS/MHz, 300 MIPS (Dhrystone 2.1) in a typical 0.13 μm process	Targets a performance range of Dhrystone 400 to 1200 MIPS
Code Density	High code density (comparable to a 16-bit microcontroller)	High code density	High code density
Die size on silicon	Small die size portable to 0.25 μm, 0.18 μm and 0.13 μm versions	Die Size 4.2 mm^2 in ARM940T. Portable to latest 0.18 μm, 0.15 μm, 0.13 μm silicon processes. Frequency 185 MHz at 0.18 μ in ARM 940T.	0.13 μm foundry processes deliver 350 to 500+ MHz in worst case and over 1 GHz on next-generation 0.1μm processes
Memory Coupling (Section 6.3)	No tight coupling	No tight coupling	
Power Performance	Very low power consumption	Very low power consumption. 940 T power 0.8 μW/MHz on 0.18 μ silicon foundry generic process. Worst case: 1.62 V, 125°C, and slow silicon. Typical: 1.8 V, 25°C, nominal silicon	Optimum power efficiency, single-issue operation with out-of-order completion to minimize gate count, consuming less than 0.4 μW/MHz on 0.13 μm foundry processes
Bus Interface	AHB	Single 32-bit AMBA bus interface	None

5. An enhancement of v5TE architecture is ARMv5TEJ architecture (introduced in 2000). It incorporates Jazelle Java execution accelerator technology for Java. This provides significantly higher Java codes execution by 8x performance than a software-based Java-Virtual-Machine (JVM). There is an 80% reduction in power consumption compared to non Java-accelerated core. This functionality gives platform developers a feature that the Java codes as well as OS applications can run on a single processor in an SoC or embedded system.
6. An enhancement of v5TEJ architecture is with ARMv6 architecture (first implementation 2002), used in ARM11 microarchitecture. It has SIMD (Single instruction multiple data) extensions, optimized for applications including video and audio CODECs. SIMD execution performance is enhanced by 4x.

Exemplary Other High Performance Processors Intel XScale and StrongARM SA-110, TI OMAP and MIPS R5000 are other examples of high performance 32 and 32/64-bit processors. These have also been used in many applications in embedded systems.

Some processors are specially dedicated to a particular performance. For example, X10 family network processor delivers 10 Gbps port performance for IPv6 (broadband Internet). DSPs with high performances are SHARC, Tiger SHARC and TMS 64x described in following subsections.

2.3.4 SHARC

SHARC is processor architecture from Analog Devices. SHARC stands for super Harvard architecture single chip computer. Figure 2.19 shows the buses, ALUs, registers and memory in SHARC architecture.

SHARC is used in a large number of DSP applications. It has controlled power dissipation in floating point ALU. Different SHARCs can be linked by serial communication between them.

SHARC has following features:

1. SHARC has 32-bit address space for accessing 16 GB or 20 GB or 24 GB as per the word size configuration in the memory. For 32-bit word size external memory configuration, addressable space is 16 GB.
2. SHARC provides for two word size configurations—32-bit and 48-bit.
3. SHARC has two full sets of 16 general-purpose registers. Therefore, context switching is fast. It thus enables multitasking OS and multithreading in programs easily.
4. Registers are called R0 to R15 or F0 o F15 depneding upon whether there are used for integer operation configuration or floating point configuration.
5. The main registers are of 32-bit. A few registers are of 48 bits so that they may also be accessed as a pair of 16-bit and 32-bit registers.
6. SHARC provides for a large ON chip memory of 1 MB. It has program memory and data memory Harvard architecture in ON chip memory.
7. SHARC also provides for external OFF chip memory.
8. OFF chip as well as ON-chip memory can be configured for 32-bit or 48-bit words.
9. SHARC architecture allows program memory configurable for program memory and data memory sections.

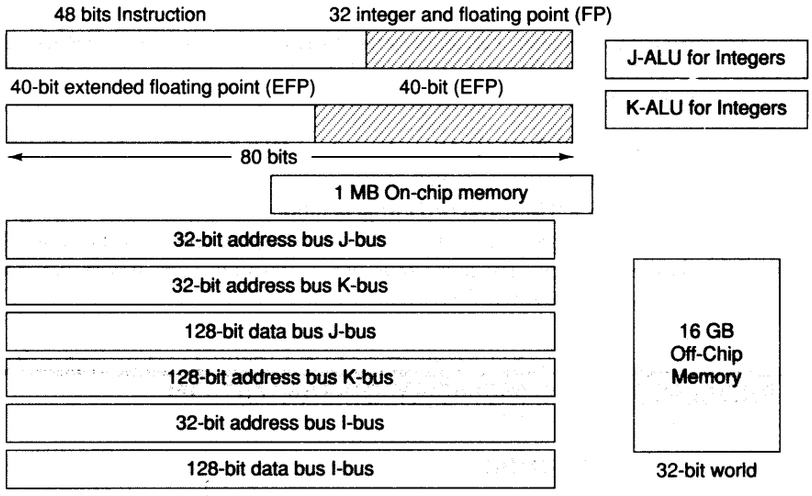


Fig. 2.20 Buses, ALUs, registers and memory in SHARC architecture

10. SHARC has instruction word of 48 bits and 32-bit data word for integer and floating point operations and 40-bit extended floating point. SHARC functions as a VLIW (very large instruction word) processor. The word size is 48-bit for instructions, 32-bit for integers and standard floating-point (FP), and 40-bit for extended floating-point (EFP). Smaller 16 or 8-bit must also store as full 32-bit data. Therefore, the big endian or little endian data alignment is not considered during processing
11. SHARC also provides instructions for saturation integer operations. For example, the integer after operation should limit to a maximum value. These instructions are required in graphic processing.
12. SHARC permits parallel operations. It supports processing instruction level parallelism as well as memory access parallelism. Therefore, there can be multiple data accesses in a single instruction.

TigerSHARC **TigerSHARC** is a highest performance density family of processors from Analog Devices. The architecture provides precision high-performance integrated circuits used in analog and digital signal processing applications. A version of TigerSHARC is TigerSHARC ADSP-TS201.

TigerSHARC is designed for multiprocessing applications and for peak performance greater than BFLOPS (billion floating-point operations per second). MultipleTigerSHARCs can connect by serial communication at .1 GBps. ADSP-TS203SABP-050 processor processes using 250 MHz clock and on chip memory of 6 M bits and operates at 1.2 V/3.3 V. Low voltage design helps in processing with little power dissipation. Analog Devices TigerSHARCs have the highest performance per watt. A TigerSHARC version has 24 M bits ON-chip memory. TigerSHARC is available as the IP core also so that new applications with the core can be developed.

TigerSHARC finds applications in software baseband processing, 3G WCDMA baseband communication, cellular base stations and 14 Mbps HSDPA (High Speed Data Packets Access) networks for packet-based multimedia contents.

2.3.5 DSP

Advanced signal processor circuits consisting of MAC (Multiply and Accumulate) unit at a DSP provides fast multiplication of two operands and accumulates results at a single address. It computes fast an expression such as the following, $y_n = \Sigma (a_i.x_{n-i})$, where the sum is made for i = 0, 1, 2, ..., N-1. Here i, n and N are the integers, a is a coefficient, x is independent variable or an input element and y is the dependent variable or an output element.

DSP processors invariably have Harvard architecture. Caches are also organized in Harvard architecture (separate I-cache and D-Cache).

Architecture of Digital Signal Processor The architecture of a DSP can be understood by considering an exemplary DSP of TMSC64x™ DSP generation.

The main structural units in a TMSC64x™ DSP generation and their functions are given in Table 2.4. Figure 2.21 shows the interconnections between twenty-five structural units by a block diagram for processor structure. Table 2.5 gives the additional structural units and the functions in the processors, TMS320C64x64™ VelociTI™, which is a VLIW architecture Extension.

2.4 PROCESSOR AND MEMORY ORGANIZATION

2.4.1 Processor Organization

Figure 2.22 shows a simple representation of organization of processor and memory in a system. The memory and IO devices interface the processor using buses. Figure 2.16 showed a detailed block diagram for internal

Table 2.4 Structural units and functions of processor in a DSP core

Structural Units in Core	*Functions*
Basic units	MDR, internal bus, data bus, address bus, control bus, bus interface unit, instruction fetch register, instruction decoder, control unit, instruction cache, data cache, multistage pipeline processing, multiline superscalar processing for processing speed higher than one instruction per clock cycle, program counter similar to Table 2.2.
Instruction dispatch	For dispatch of instructions to the appropriate units.
Control register	Control registers associated with the control unit of the processor.
Registers emulation unit	Emulation
Register File A	Set of on-chip registers used during processing instructions in data path 1. These are named A0... A 15 and A16 ...A31. A register file is a file that associates with a unit such as ALU or FLPU.
Register File B	Set of on-chip registers used during processing instructions in data path 2. These are named A0... A 15 and A16 ...A31.
Prefetch unit	For fetching eight 32-bit instructions at each cycle.
Processing unit	Two multipliers and six arithmetical units, highly orthogonal, compiler and assembly optimizer, execution resources.
Arithmetic logical subunit	Subunit to execute arithmetical or logical instruction according to current instruction fetched at IR.
Auxiliary Logic subunit	A subunit used during subtraction. [Finds 2's complement before addition and then adds in order to subtract]
Multiplier subunit	Multiply
Floating Point processing (FLP) subunit	Subunit in C67x ™ distinct from the ALU and performs FLP operations.
Assembly Optimizer	Optimizer for assembled codes
C compiler	Highly efficient compilation

Table 2.5 Additional structural units and functions of processors in TMS320C64x64™ VelociTI ™ VLIW architecture extension

Structural Unit in Core	*Functions*
Packed data processing	8-bit or 16-bit data packed and processed as 32-bit data
Parallel execution MAC units	Quad 16-bit MAC/Octal 8-bit MAC [Table 2.2]
Special instructions	Broadband and image processing using VLIWs
Level 2 cache	Enhances performance of each fetch cycle
Instruction packing unit	Instructions packed as VLIW, which executes in parallel without in between halts

units of processor and showed the buses. A processor has an ALU. A processor circuit does sequential operations and a clock guides these. A processor has the program counter and stack pointer, which point to the instruction to be fetched and top of the data pushed into the stack, respectively. Certain processors have on-chip memory

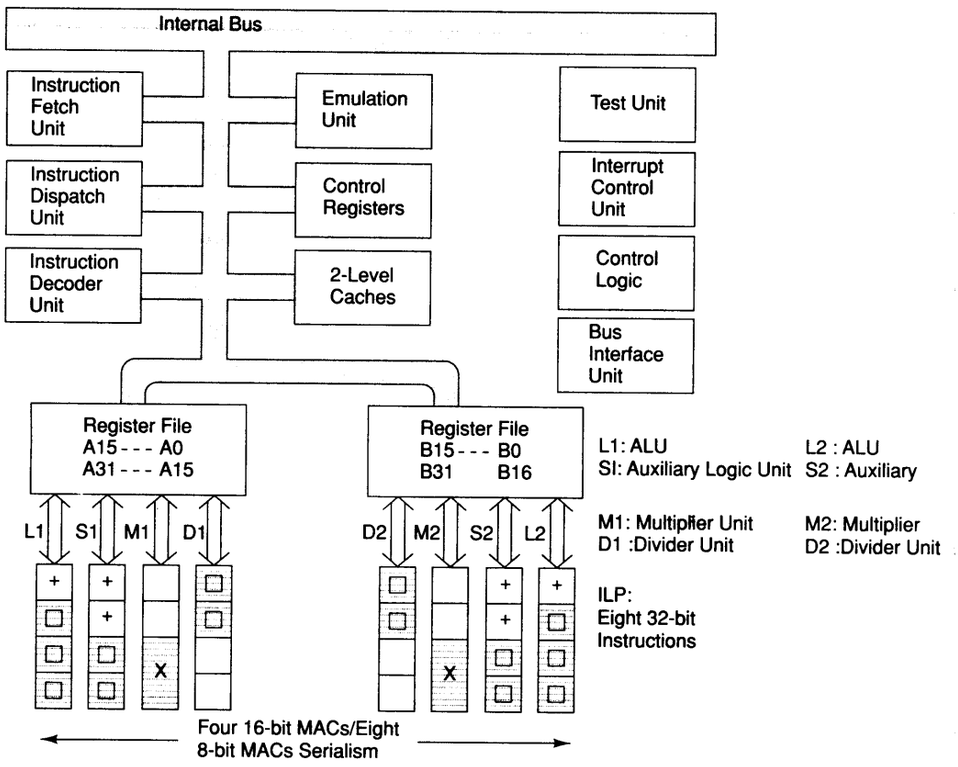


Fig. 2.21 Core and special structure units in DSP, TMS320C64x DSP
Note: Floating Point Units present in C67x

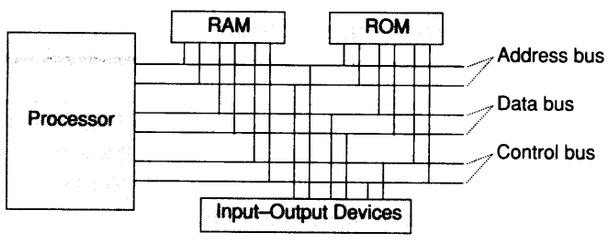


Fig. 2.22 A simple view of organization of processor, buses and memory in a system

management unit (MMU). A processor generally has general-purpose registers. Registers organize onto a common internal bus of the processor. A register is of 32, 16 or 8 bits depending on whether the ALU performs at an instance 32- or 16- or 8-bit operation.

A processor may have CISC (Complex Instruction Set Computer) or RISC (Reduced Instruction Set Computer) architecture. A CISC has the ability to process complex arithmetic and logic as well as other

instructions and processes complex data sets using fewer registers, as it provides for a large number of addressing modes. An RISC executes simpler instructions and in a single cycle per instruction. New RISC processors, such as ARM 7 and ARM9, also provide for a few most useful CISC instructions also. CISC converges to an RISC implementation because most instructions are hardwired and implement in a single clock cycle.

A processor provides for the inputs for external interrupts so that the external circuits can send the interrupt signals (Section 2.2.4). The processor may possess an internal interrupt controller (handler) to program service routine priorities and to allocate vector addresses. The internal interrupt controller is of great help in most applications.

A processor may provide for bit manipulation instructions. These instructions help in easy manipulation of bits at the ports and memory addresses. Certain processors possess FLPU and FRS units that perform floating-point operations fast. These permit higher computational capabilities in the processor; they are essential for signal processing and sophisticated control applications.

Certain processors provide for direct memory access (DMA) controller with multiple channels on chip. When there are a number of I/O devices and an I/O device needs to access a multibyte data set fast, the system memory on-chip DMA controller is of great help. Section 4.8 will describe the DMA in detail.

Table 2.6 lists the nineteen features for the CISC family of microcontrollers and microprocessors.

Table 2.6 Features in four CISC microcontroller and processor families

Capability	*Intel 8051 and Intel 8751*	*Motorola M68HC11E2 E2*	*Intel 80196KC*	*Intel Pentium*
Processor instruction cycle in microseconds (typical)	1	0.5	0.5	0.001[1]
Iinternal bus width in bits	8	8	16	64
CISC or RISC architecture	CISC	CISC	CISC	CISC with RISC feature[2]
Program counter bits with reset value	16 (0x0000)	16 [(0xFFFE)]	16 (0x2080)	32[3] (0xFFFF FFFF)
Stack pointer bits with initial reset value in case a processor defines these	8 (0x07)	16	16	32[3]
Atomic operations unit	No	No	No	No
Pipeline and super-scalar architecture	No	No	No	Yes
On-chip RAM and/or register file bytes[4]	128 and 128 RAM	512 RAM	256 and 232	No
Instruction cache	No	No	No	8 kB[5]
Data cache	No	No	No	8 kB[5]
Program memory EPROM/EEPROM	4 k	8 k	8 k	No
Program memory capacity in bytes	64 k[6]	64 k	64 k	4 GB
Data/ stack memory capacity in bytes	64 k[6]	64 k	64 k	4 GB
Main memory, Harvard or Princeton architecture (Section 2.4.2)	Harvard[6]	Princeton	Princeton	Princeton
External interrupts	2	2	2	1[7]
Bit manipulation instructions	Yes	Yes	Yes	Yes
Floating point processor	No	No	No	Yes

(Contd)

Capability	*Intel 8051 and Intel 8751*	*Motorola M68HC11E2 E2*	*Intel 80196KC*	*Intel Pentium*
Internal Interrupt controller	Yes	Yes	Yes	No
DMA controller channels	No	No	1 (PTS)[8]	4
On-Chip MMU	No	No	No	Yes

[1] It is maximum time in a typical Pentium 1 GHz version.
[2] Single clock-cycle hardwired implementation for most instructions implement like a RISC
[3] Stack Pointer ESP 32 bits together with the Stack Segment ES 16 bits point to physical stack address at the memory. It equals ES × 0x10000 + ESP.
[4] This is in standard version. In other versios, it may be different.
[5] This is in a typical version
[6] Program and data memory spaces are separate in Intel 8051 family members. It is common in others.
[7] Using the INTR pin and external programmable interrupt controller, up to 256 external interrupts can be handled
[8] PTS means there is a Peripheral Transactions Server providing a DMA-like feature.

Table 2.6 shows the memory addresses in hexadecimal. Thus 0x10000 means a hexadecimal memory address 10000; 0x100FF means hexadecimal memory address 100FF. The same is the convention in C. It helps in distinguishing a decimal number from hexadecimal number.

2.4.2 Memory Organization

The memory system (consisting of various units) acts as a storage receptacle for data and programs. Most systems have two types of memory—*read-only memory* (ROM) and *random-access memory* (RAM). A flash memory functions as the ROM. Examples of uses of flash are mobile phone, mobile-computer and digital camera.

Read Only Memory As its name suggests, contents of the ROM does not modify during running of computer or on power off but may be read. In general, the ROM is used to hold a program that is executed automatically by the system every time it is turned on or reset. This program is called bootstrap, or boot loader, which instructs the system to load its operating system from its hard disk or other I/O storage device. The name of this program comes from the idea that the system is "pulling itself up by its own bootstraps" by executing a program that tells it how to load its operating system. An example of ROM is as follows: A system has ROM unit(s) for the bootstrap program(s), basic input–output system (BIOS) program(s) and vector addresses of the interrupts (Section 4.4.1).

Random Access Memory Random-access memory, on the other hand, can be both read and written, and is used to hold the programs, operating system and data required by the system. For example, a mobile phone has 128 kB or 256 kB of RAM to hold the stack and temporary variables of the programs operating system and data. RAM is generally volatile, meaning that it does not retain the data stored in it when the system's power is turned off. Any data that needs to be stored while the system power is off must be written to a permanent storage device, such as flash memory or hard disk.

Addresses Memory (both RAM and ROM) is divided into a set of storage locations, each of which can hold 1-byte (8 bits) of data. The storage locations are numbered, and an assigned number is called *address*. It defines in a memory of system which location the processor wants to reference at a given instance. One of the important characteristics of a computer system is the width of the address lines (bus) it uses, which limits the amount of memory that the processor can address. Most current computers use either 32-bit or 64-bit addresses,

allowing them to access either 2^{32} or 2^{64} bytes of memory. Assume that an IBM PC has 1 MB memory (1024 × 1024 bytes). Its bootstrap program and BIOS ROM addresses are between 15×2^{16} (≡ 0xF0000) and $2^{20} - 1$ (≡ 0xFFFFF). RAM addresses are between 1×2^{16} (≡ 0x10000) and $15 \times 2^{16} - 1$ (≡ 0xEFFFF).

Random Access Model of Memory A simple model for RAM and ROM both is the random-access model of memory when all memory operations take the same amount of time independent of the address of byte or word in memory. Assume that the memory system will support two operations: load (read operation into processor from memory) and store (read operation from processor into memory). The random access model states as follows: From the memory, a data byte, a word, a double word, or a quad word may be accessed from or at any addressable location, and a similar process is used to access from all locations. There is equal access time for a read or write that is independent of a memory address location. This mode differs from another model, called serial access model.

Store and Load (Write and Read) Instructions Most high performance organizations allow more than 1-byte of memory (generally four bytes) to be loaded or stored at one time. Generally, a load or store operation operates on a quantity of data equal to the system's bus width, and the address sent to the memory system specifies the location of the lowest-addressed byte of data word(s) to be loaded or stored. Each instruction mostly has the opcode followed by operands. Store operations need two operands, a value to be stored and the address in which that the value should be stored. They place the specified value in the memory location specified by the address.

Load operations need an operand that specifies the address containing the value to be loaded and return (fetch) the contents of that memory location into their destination (register), which is specified by another operand.

Using this model, the memory can be thought of as functioning similar to a large sheet of lined paper, where each line on the page represents a 1-byte storage location. To write (store) a value into the memory, we count down from the top of the page until we reach the line specified by the address, erase the value written on the line and write in the new value. To read (load) a value, we count down from the top of the page until we reach the line specified by the address, and read the value written on that line.

Alignment of Multibyte Store and Load in a Memory Organization Some memory organization requires loads and stores to be "aligned". Assume that a 4-byte word has been aligned at address 0x000C or 0x1000, which is a multiple of 4. This simplifies the organization of the memory system as follows:

When a memory organization require loads and stores to be "aligned," it means that the address of a memory reference must be a multiple of the size of the data being loaded or stored, so a 4-byte load must have an address that is a multiple of 4, an 8-byte store must have an address that is a multiple of 8, and so on. Other systems allow unaligned loads and stores, but take significantly longer to complete such operations than aligned loads.

ARM processor memory addresses are aligned either in multiples of four or two or one byte addresses. ARM permits three data types: four bytes word or two byte half word or 1-byte word, which stores at addresses in multiple of 4 or 2 or 1, respectively.

Example 2.9

(a) Assume that a given memory organization require loads and stores to be "aligned". Then a 32-bit system loads or stores 32 bits (4 bytes) of data with each operation into the 4 bytes that start with the operation's address, so a load from location 0x424 would return a 32-bit word containing the bytes in locations 0x0424, 0x0425, 0x0426 ad 0x0427.

(b) Assume that a given organization require loads and stores to be not aligned. A 32-bit system loads or stores 32 bits (4 bytes) of data with each operation into the 4 bytes that start with the operation's address, so a load from location 0x423 would return a 32-bit word containing the bytes in location 0x0423 0x0424 0x0425 and 0x0426, as in such organizations the store or load address can be any number, not necessarily a multiple of 2 or 4.

Little Endian and Big Endian in a Memory Organization Some processor and memory organizations require little endian and other big endian aligned multiple bytes when there is store into the memory or load into the processor from memory. The ARM processor permits programming at the start and enables a programmer to define one of two possible word-alignments, little endian or big endian, at the beginning. It is important to know how organization orders the bytes written at the memory.

(a) In a little-endian system, the least-significant (smallest value) byte (8-bit) of a word (of 16 or 32-bit) is written into the lowest-addressed byte, and the other bytes are written in increasing order of significance.

(b) In a big-endian system, the byte order is reversed, with the most significant byte being written into the byte with the lowest address. The other bytes are written in decreasing order of significance.

Example 2.10

1. Two different ordering schemes are used in modem computers: *little endian* and *big endian*. Assume that a word of 32 bits is 0x90ABCDEF, and the address where the word stores when written is 0x1000. The following shows an example of how a little-endian system and a big-endian system would write a 32-bit (4-byte) data word to address 0x1000.

Little-endian system and a big-endian system

Address	*0x1000*	*0x1001*	*0x1002*	*0x1003*
Little Endian	EF	CD	AB	90
Big Endian	90	AB	CD	EF

In general, programmers do not need to know the endianness of the system they are working on, except when the same memory location is accessed using loads and stores of different lengths. For example, if a 1-byte store of 0 into location 0x1000 was performed on the 32-bit systems in Example 2.10, a subsequent 32-bit load from 0x1000 would return 0x90ABCD00 on the little-endian system and 0x00ABCDEF on the big-endian system. Endianness is often an issue when transmitting data between different computer systems, as big-endian and little-endian computer systems will interpret the same sequence of bytes as different words of data. To get around this problem, the data must be processed to convert it to the endianness of the computer that will read it.

Figures 2.10 and 11 described the memory, processor and IO units organized on the buses. It can be safely concluded that the memory organization has a tremendous impact on computer system performance and is often the limiting factor on how quickly an application executes. Both bandwidth (how much data can be loaded or stored in a given amount of time) and latency (how long a particular memory operation takes to complete) are critical to application performance.

Other important issues in memory system design include protection (preventing different programs from accessing each other's data) and how the memory system interacts with the IO system.

There may be on-chip memories as RAM and/or register files, windows, caches and ROM in a microprocessor.

The caches are the integral parts of the memory-organization within a system. The software designer should enable the use of caches by an appropriate instruction, to obtain greater performance during the run of a section of a program, while simultaneously disabling the remaining sections in order to reduce the power dissipation and minimize energy requirements. Hardware designers should select a processor with multiway cache units so that only that part of a cache unit gets activated that has the data necessary to execute a subset of instructions. This also reduces power dissipation.

Processor Memory Organization: Princeton Architecture Figure 2.23(a) shows processor and memory organization in Princeton architecture. 80x86 processors and ARM7 have Princeton architecture for main memory. Vectors, pointers, variables, program segments and memory blocks for data and stacks have different addresses in the program in Princeton memory architecture.

Processor Memory Organization: Harvard Architecture Figure 2.23(b) shows processor and memory organization in Harvard architecture. A processor having Harvard main-memory architecture has distinct address spaces, control signal(s), processor instructions, and data paths for the bytes for data and for program. (The 8051-family microcontrollers have Harvard architecture.)

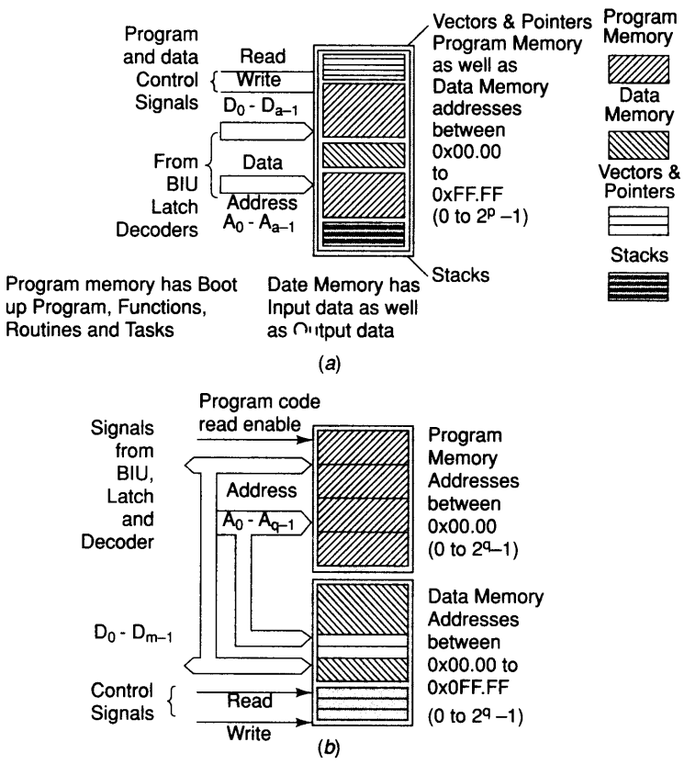


Fig. 2.23 (*a*) Processor and memory organization in Princeton architecture
(*b*) Processor and memory organization in Harvard architecture

Harvard architecture helps in handling streams of data that are required to be accessed in cases of single instruction multiple data type instructions and DSP instructions. Separate data buses ensure simultaneous accesses for instructions and data. Program segments and memory blocks for data and stacks have separate sets of addresses. Control signals and read-write instructions are also separate for accessing the program memory and data memory.

It must be remembered when coding in assembly and when organizing the main memories of certain processors, that their memory organization may be Harvard architecture. Program memory and data memory have separate set of addresses and have separate instructions for area accesses. A processor having Harvard architecture is needed for access to streams of data. Examples are (i) single instruction multiple data type instructions and (ii) DSP instructions.

For example, consider a DSP computation of the following expression in a 'Finite Impulse Response (FIR) filter'. An n-th filtered output sequence, $y_n = \Sigma(a_i.x_n - i)$, where the sum is made for i = 0, 1, 2, ..., N–1. Here i, n and N and the integers. If N = 10, then for each value of y, first one of the 10 coefficients, a_i, and one of the 10 input sequences, x, are multiplied and then the summation is done. The total computations for all 10 values of n will need 100 multiplications and 100 summations. Storing and accessing the coefficients from a separate set of memory-addresses in a separate memory will allow fast access by using a separate set of buses.

2.5 INSTRUCTION-LEVEL PARALLELISM

Several instructions can execute in parallel. Two or more instructions can execute in parallel as well as in sequence in pipelines. In the instruction level parallelism (ILP), two parallel pipelines in a processor and two instructions I_n and I_{n+1} execute in parallel at separate execution units. Figure 2.24 shows instruction-level parallelism in pentium processor.

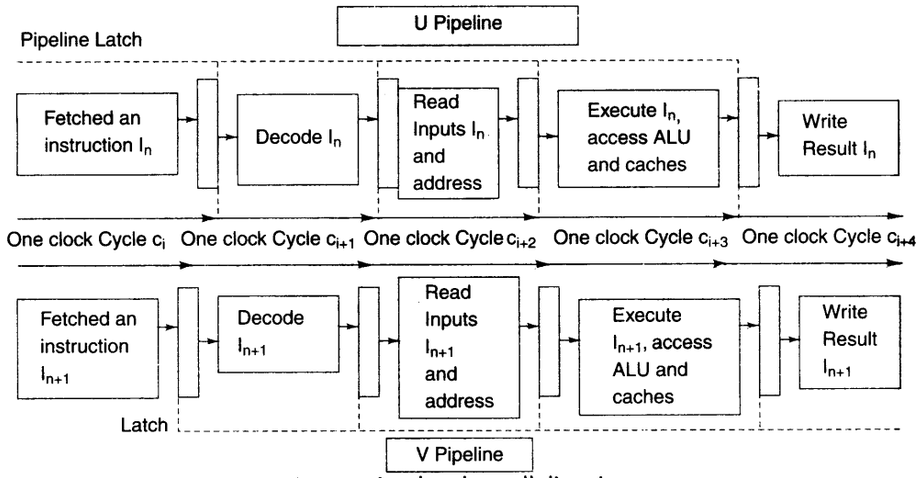


Fig. 2.24 Instruction level parallelism in a processor

2.5.1 Pipelined and Superscalar Units

High processor performance is required in many cases. For example, real-time signal processing. Pipelining and superscalar operations have now become essential. The hardware designer selects the processor as per the

required MIPS or MFLOPS performance. [A multi-processor system (Section 6.4.1) will be needed for very high performance requirements in mobile phones, digital camera, speech processing and video systems.]

Advanced processing units include instruction pipelining unit, which improves performance by processing instructions in multiple stages and the parallel units for superscalar execution, which improves performance on execution of two or more instructions in parallel execution units.

How do pipeline and superscalar units give such higher performance? Let us look at Example 2.11.

Example 2.11

Pipeline and Superscalar Execution

Step 1: Let us assume that the processor instruction cycle time is 0.02 μs (at 50 MHz operation) and that the processor executes an instruction in one clock cycle. The processor performance expected without advanced processing units will be 50 MIPS.

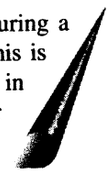
Step 2: Assume there is a three-stage pipeline as in ARM7. Let us, for the moment, ignore the effect of branching (called *branch penalty*). Three instructions will process in three clock cycles, but each clock cycle period can be made = 1/3 of the earlier period, as the division of processing units in stages divides the circuit also. The maximum expected performance of the processor without superscalar but with pipeline will be = 150 MIPS.

Step 3: Assume there is a two-line superscalar. Let us ignore the effects of unaligned data (*data dependency penalty*). Six instructions can process in single clock cycle with the three-stage pipeline and two superscalar units. The maximum performance will now be six times the processor cycle time, 300 MIPS.

We now explain the two terms used: branch penalty and data dependency penalty.

Branch penalty: If a branching instruction is encountered at a multistage pipeline, then the instructions executed in part at the preceding stages become redundant. These instructions have to be executed in full again later on after completion of the loop or return from a routine. The time required for re-processing these is called branch penalty.

Data dependency penalty: Assume that there are two instructions in two execution lines during a superscalar operation. Further, that one instruction depends on the data output of another. This is known as improper alignment. Thus, the two instructions are not aligned before putting them in separate lines. One instruction will now have to wait and cannot proceed further till the other instruction is executed. The waiting time is the data dependency penalty.



Superscalar processors possess hardware to extract instruction-level parallelism from sequential programs and possess hardware to efficiently take care of the collisions in execution unit and of data and control hazards.

During each cycle, the instruction issue logic of a superscalar processor examines the instructions in the sequential program to determine which instructions may be issued on that cycle. If enough instruction-level parallelism exists within a program, a superscalar processor can execute one instruction per execution unit per cycle, even if the program was originally compiled for execution on a processor that could only execute one instruction per cycle.

SHARC supports instruction level parallelism. The SHARC processor has instructions that are combined as single instruction word. [SHARC also supports memory parallelism; its processor has a modified Harvard structure called super Harvard structure (Figure 2.20). Multiple data can be fetched in a single instruction.]

ILP capability is one of the greatest advantages of superscalar processors and is the reason why virtually all high performance CPUs are superscalar processors. Superscalar processors can run programs that were originally compiled for purely sequential processors, and they can achieve better performance on these programs than processors that are incapable of exploiting the ILP.

Thus, users who work on new systems containing superscalar CPUs can install their old programs on those systems and see better performance on those programs than was possible on their old systems.

The use of high performance processor ICs and cores in embedded systems providing billion operations per second has become feasible due to the great advances in VLSI and in ILP and multi core processor design technology.

2.6 PERFORMANCE METRICS

Sophisticated embedded systems for high computing performance applications needs optimized use of resources, power, caches and memory. The following are the processor performance metrics:

1. (a) High MIPS, (b) high MFLOPS and (c) high *Dhrystone benchmark* program-based MIPS
2. Optimized compiler unit performance in the processor.

The above metrics are provided by the latest innovatively designed processors. A high-performance processor combines capabilities with optimized use of resources, power, caches, and memory.

A benchmarking program is called Dhrystone, developed in 1984 by Reinhold P. Weicker. It measures the performance of a processor for processing integers and strings (characters) both. It uses a benchmark program available in C, Pascal or Java. It benchmarks a CPU and not the performance of IO or OS calls. Dhrystones per second is the metric used to measure the number of times the program can run in a second. 1 MIPS = 1757 Dhrystone/s. [Why? VAX11/780, which executed 1 MIPS, ran the Dhrystone benchmark program 1757 times (refer to http://www.webopedia.com/ TERM/D/Dhrystone.html.)]

There is EDN Embedded Benchmark Consortium (EEMBC) [EDN is a group that publishes the International magazine EDN, which is dedicated to Embedded System information. Refer to http://www.e-insite.net/edmag/]. EEMBC proposed five-benchmark program suites for 5 different areas of applications of embedded systems: (a) Telecommunications, (b) Consumer Electronics, (c) Automotive and Industrial Electronics, (d) Consumer Electronics, and (e) Office Automation. These program suites are also used for measuring and comparing embedded system processor performances.

Different systems require different processor performance in terms of processing speed. A hardware designer takes these into view and selects an optimum performance-giving processor.

2.7 MEMORY-TYPES, MEMORY-MAPS AND ADDRESSES

2.7.1 Memory in a System

Section 1.3.5 introduced the memory in a system. A simple credit–debit transaction card may require just 2 kB of memory. On the other hand, a smart card for secure transactions when embedding a Java program for cryptographic functions may require 32 kB (typical value) memory. A complex embedded system may need huge memory.

The following subsections explain and look at certain important aspects of the memory. The various memory are described from the point of view of an embedded systems hardware or software designer.

ROM: Its Uses, Forms and Variants ROM non-volatility is a most important asset and it is extremely useful to embed codes and data in a system. ROM is a loosely used term. For a hardware designer, it may mean masked ROM, PROM, OTP-ROM, EPROM and EEPROM. In a strict sense, ROM means a masked ROM

made at a foundry from the programmer's ROM image file (Section 1.4.1). ROM that embeds the software or an application logic circuit is in one of the three forms: masked ROM, PROM and EPROM. When ROM is to be programmed during runtime and is to hold the processed result, either an EEPROM or flash memory is used.

(i) Masked ROM A masked ROM is built from a circuit that has r inputs (A_0 to A_{r-1}) and 8 outputs (D_0 to D_7). [Byte storing at an address is most common.] The circuit for masked ROM is one of a set of 2^r combinational circuits. Appropriate masking gives the desired set of outputs at each combinational circuit. Certain links fuse and others that are masked do not fuse. [A combination circuit is a circuit made up of logic gates with a distinct set of output logic states during distinct input logic states. It has a distinct truth table for r inputs x 8 outputs. As soon as the inputs change (or withdraw), the output also changes in this circuit.].

The embedded software designer (after thorough testing and debugging) provides to a manufacturing foundry a file having a table of desired output bits for the various combinations of the input address bits. A program called *locator* creates this table. The manufacturer prepares the programming masks and then programs the ROM at a foundry. This ROM is returned to the system manufacturer.

Normally, one time masking charge could be very high. Generally, therefore, a system manufacturer will place the order, and the manufacturing foundry will accept the order for a minimum of 1000 pieces. The ROM is a cost effective solution to a bulk user of ROMs for the manufacture of embedded systems. An embedded system manufacturer using a masked ROM does not have to use a device programmer (ROM burner) each time a system ROM is made using EPROM or PROM or flash.

(ii) EPROM, E^2PROM and OTP ROM Special versions of ROM can be programmed at the designer's or manufacturer's site for an embedded system with the help of a device programmer. One version is EPROM. It is an ultraviolet ray *erasable* and device programmer *Programmable Read Only Memory*. Erasing the device means restoring 1 at each bit in the cell arrays at each ROM address. Another version is E^2PROM (EEPROM). It is an *Electrically Erasable,* and *Programmable Read Only Memory*. Examples of EPROM and EEPROM are 2732, a 4 kB EPROM, 28F256, a 32 kB EEPROM and 28F001 is 512 K × 16-bit EEPROM.

EEPROM erasing during an application-program run is done by sending all eight data bus bits as 1s for the write in the presence of inputs of V_{pp}, called programming voltage and short duration write pulse. Sending ls and 0s in the byte by a write instruction results in EEPROM programming during a program run. Erasing of a byte must precede the write. The processor with the system program can do erasing and writing, as it is similar to the writing in a RAM. What then, is the difference between the EEPROM and RAM? The difference is that in RAM, the read and write timing cycles are identical. Here, the write cycle has to be longer than in case of RAM, and it must succeed the erase of the byte by writing 0xFF. Further, an addition voltage V_{pp} signal is needed when erase and write occurs to the EEPROM The number of times an EEPROM can be written is one million times plus. There is no limit for RAM, and a practically infinite number of writes is possible without first writing 1s in RAM.

Flash memory is a form of EEPROM in which a sector of bytes can be erased in a flash (very short duration corresponding to a single clock cycle). [Lately flashes of even ~3 V form and of capacity 2, 4 and 8 GB have become available even in card or stick form, which enables insertion in camera or pocket computer.] A sector can be from 256 B to 16 kB. The advantage over EEPROM is that the erasing of many bytes simultaneously saves time in each erase cycle that precedes the write cycles. The disadvantage is that once a sector is erased, each byte writes into it again one by one, and that takes too long a time. A new version of flash is **boot back flash**. A sector is reserved to store once only at the time of first boot. Later on it is protected from any further erase. In other words, it has an OTP sector also that can be used to store ROM images like in a ROM. Nowadays, flash has replaced EPROM in the systems.

PROM (an OTP ROM, a one time device programmer) is another form. A PROM once written is not erasable.

(iii) Uses of ROM or EEPROM or Flash Figure 1.5 showed what a ROM embeds—program codes for various tasks, interrupt service routines, operating system kernel, initialization (bootstrap program and data) and the standard data or table or constant strings.

An EEPROM is usable by erasing over one million times. It can be erased during runtime itself. Flash memory is usable about 10,000 times for repeated erasing followed by programming during the runtime. The PROM is written only once by a device programmer or the first system run.

Three examples of EEPROM memory applications are as follows. **(i)** Storing current date and time in a machine. **(ii)** Storing port statuses. **(iii)** Storing driving, malfunctions and failure history in an automobile for use by mechanics later on.

Three examples of flash memory applications are as follows. **(i)** Storing pictures in a digital camera. **(ii)** Storing voice compressed form in a voice recorder. [Recall of prerecorded message in a phone.] **(iii)** Storing messages and contacts in a mobile phone.

Examples of use of an OTP ROM are as follows. **(i)** Smart card identity number and user's personal information. **(ii)** Storing boot programs and initial data like a pictogram displaying a seal or monogram. **(iii)** ATM card or credit card or identity card. Once the various details are written at the bank and handed over to the account holder, there is no modification possible in the embedded PROM at the card. Just as a paper holds information permanently once written or printed, so also does a PROM.

A flash or PROM or ROM is not only used for program and data storage, but also for obtaining the preprogrammed logic outputs and output sequences for the given sets and sequences of inputs. [Inputs are given analogous to an address signal by the processor and outputs are obtained analogous to those obtained during a processor read cycle.] Assume that there are 8 inputs (r = 8). The truth table for it will have 256 combinations. 8 x 8 ROM can be programmed to generate 256 sets of 8-bit outputs for each combination. Examples of applications of preprogrammed logic outputs are as follows.

1. Used to hold language-specific bits for the fonts corresponding to each character in a printer.
2. Used to hold the image bits for a display. A pictogram generates from these bits. A ROM is used in a display circuit. It stores the bytes for the full bit-image corresponding to the pixels for a pictogram. Sequential changes at the inputs repeatedly generate the full pictogram.
3. In a CISC a control ROM at a micro-programmed unit is used. It stores sets of microinstructions for each processor instruction. Each set of microinstructions is stored in a sequence such that it specifies a set of signals for the various fetch and executing unit during execution of an instruction fetched from the memory.

RAM A system designer considers RAM devices of eight forms. These forms are 'SRAM', 'DRAM', 'NVRAM', 'EDO RAM', 'SDRAM', 'RDRAM', Parameterized Distributed RAM and Parameterized Block RAM.

(i) Uses of RAM RAM stores the variables during a program run and stores the stack (Section 1.3.5). It stores input and output buffers, for example, of speech or image. It can also store the application program and data when the ROM image is stored in a compressed format in an embedded system and decompression is done before the actual run of the system.

1. SRAM is used most commonly for designning caches and in embedded systems and microcontrollers.
2. DRAM is used mostly in high performance computers or high memory density systems.
3. EDO RAM is used in systems with buses to the devices when operating with clock rates up to 100 MHz; a zero-wait state is needed between two fetches, and there is single-cycle read or write.

4. SDRAM synchronizes the read operations and keeps the next word ready while the previous one is being fetched. This device is used when buses can fetch or send to the processor up to speed of 1 GHz.
5. RDRAM accesses in bursts the four successive words in a single fetch and thus gives above 1 GHz performance of the system.
6. Parameterized distributed RAM is the RAM distributes in various system subunits. IO buffers and transceiver subunits can have a slice of RAM each and the system stack can be at another slice. Distribution provides buffering of memory at the subunits before they are fetched and processed by the processor. It facilitates faster inputs from the IO devices than the processor system buses access the IOs using system memory.
7. Parameterised block RAM is used when a specific block of the RAM is dedicated for use by a subunit only, for example, MAC unit. A parameterized block RAM is used when an access by the system or IO or internal bus is slow compared to the processing speed of a subunit.

Different types of memory in varying capacities are available for use as per requirement. (1) Masked ROM or EPROM or flash stores the embedded software (ROM image). Masked ROM is for bulk manufacturing. (2) EPROM or EEPROM is used for testing and design stages. (3) EEPROM is used to store the results during the system program runtime. It is erased byte-by-byte and written during the system-run. It is useful to store modifiable bytes, for example, the runtime system status, time and date and telephone number. (4) Flash stores the results byte by byte during a system run after a full sector erase. (5) Flash is thus very useful when a processed image or voice is to be stored or a data set or system configuration data is to be stored, which can be upgraded as and when required. For example, a new image (after compressing and processing) can be stored and the old one erased from a sector in a single instruction cycle. (6) Boot block flash also has an OPT sector(s) to store the boot program and initial data or permanent system configuration data. It serves by storing the ROM image or its part in the OTP sector(s) and, at the same time, serves by storing as a flash in other sectors. (7) RAM is mostly used in SRAM form in a system. (8) Sophisticated systems use RAM in the form of a DRAM, EDO RAM, SDRAM or RDRAM. (9) Parameterized distributed RAM is used when the IO devices and subunits require a memory buffer and a fast write by another system. (10) Subunits like MAC when operating at fast speed use separate blocks of RAM.

2.7.2 Address Allocations in Memory

Figure 2.23 (a) showed memory addresses needed in the case of Princeton architecture in the system. Figure 2.23 (b) showed memory addresses needed in the case of Harvard architecture.

A *system memory allocation-map* is not only a reflection of addresses available to the memory blocks, and the program segments and addresses available to IO devices, but also reflects a description of the memory and IO devices in the system hardware. It maps guides to the actual presence of the memory at the various units, EPROM, PROM, ROM, EEPROM, flash Memory, SRAM (static RAM), DRAM (dynamic RAM) and IO devices. It reflects memory allocation for the programs, and data and IO operations by a *locator* program. It shows the memory blocks and ports (devices) at these addresses.

System IO devices map may be designed separately. This not only reflects the actual presence of the IO devices, but also guides the available addresses of the various device registers and port-data. [An example of a device is a timer. IO devices are the peripheral units of the system.]

The following are examples that describe memory allocation maps using the *locator*.

Example 2.12

Consider a memory map for an exemplary embedded system—a smart card needing a 2 kB memory, a 256 B RAM mainly for the stacks, EEPROM 512 B for storing the balance amount under credit or debit and the previous transaction records on the card. The memory locator or linker script program for this system to define a memory allocation map [Figure 2.25(a)] is as follows.

```
1. Memory
2. {ram : ORIGIN = 0x10000, LENGTH = 256
3. eeprom : ORIGIN = 0x20000, LENGTH = 512
4. rom : ORIGIN = 0x00000, LENGTH = 2K
5. }
```

Example 2.13

Consider a Java embedded card with software for encrypting and deciphering transactions. Assume that the system needs 32 kB ROM, RAM of 4 kB, and EEPROM 512B for storing not only the balance amount under credit or debit but also the cryptographic keys and previous transaction records on the card. So the memory locator or linker script program for this system defines memory map [Figure 2.25(b)] as follows.

Memory

```
1. { ram : ORIGIN = 0x10000, LENGTH = 4K
2.  eeprom : ORIGIN = 0x20000, LENGTH = 512
3.  rom : ORIGIN = 0x00000, LENGTH = 32K
4. }
```

One can also make the following important observation from Examples 2.12 and 2.13. There are memory address gaps between the origin of ROM, RAM and EEPROM in spite of the very small lengths of available memory. This gap is due to a design feature: the designer provides for expansion of these memories in future so no change will be needed in the interfacing decoder circuit between the memory and processor. Further, its software program has to make minimal changes. The changes will only be in length. This is because when there is no gap the origin will also change. This feature ensures that any future changes in the program code sizes and data sizes will not need change in the locator codes. One feature of a locator is also that it does not relocate the addresses of the special purpose ports that are dedicated to a particular IO task or dedicated to the device driver read and write operations.

The final step of the design process in an embedded system is that the bytes locate at the ROM from the image for the bootstrap (reset) program and data, the initialization data as well as the following standard data or table or constant strings device driver data and programs, the program codes for various tasks, interrupt service routines and operating system kernel. [The bootstrap program consists of the instructions that are executed on system reset. The bootstrap data example is for stack pointer initialization. The initialization data may be for defining initial state and system parameters. The constant strings may be for initial screen display.] There is a shadow segment in the ROM. The shadow segment has the initialization data, constant string, and the start-up codes that are copied into the RAM by a shadow segment copy program at system boot up. When a start-up code (booting) program is executed, a copy of the shadow segment from the ROM is generated in the RAM. The RAM also holds the data (intermediate and output data) and stack. *A compressed program format locates at the ROM in case of large ROM image is required for the system program. This is because decompression program plus compressed image will need less memory than the large ROM image. The start-*

up code task is to generate the decompressed program codes and store them into the RAM before system starts other programs. The processor executes all other programs subsequently by fetches from the RAM.

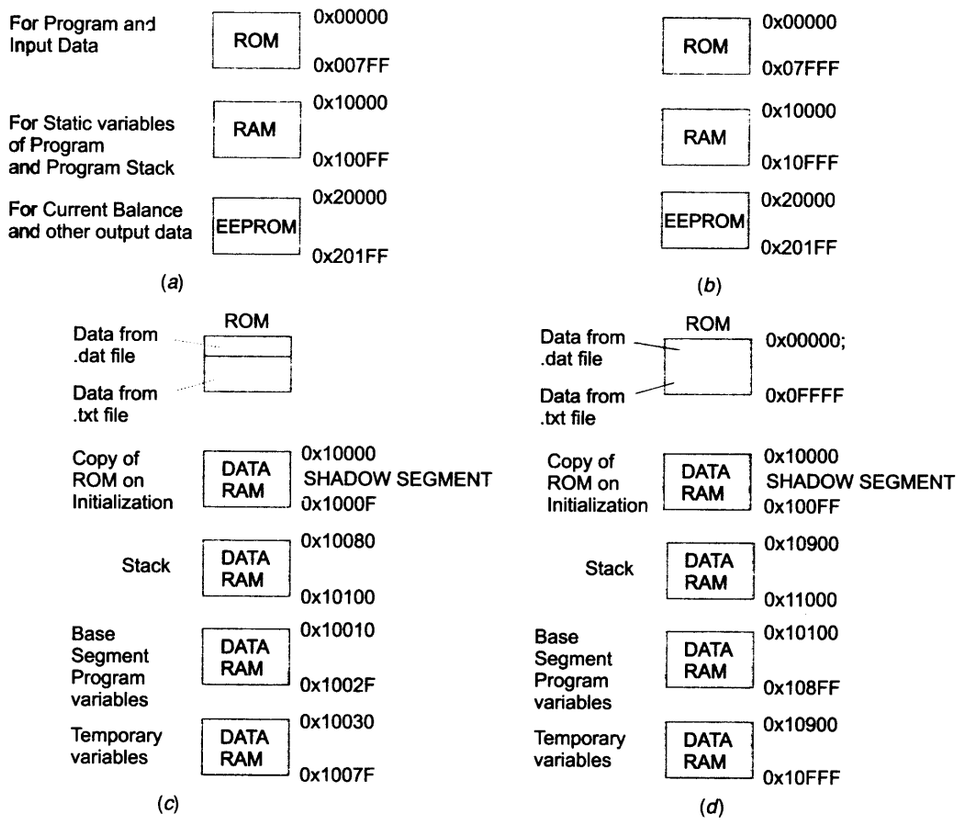


Fig. 2.25 Examples of Memory maps in embedded systems—(*a*) smart card needing 2 kB memory (*b*) Java embedded card with software for encrypting and deciphering transactions (*c*) memory map sections in a smart card (*d*) memory map sections in another smart card

A processor may have predefined memory locations for the initialization boot record. For example, in 80960, boot record consists of 12 words. The record is stored at ROM addresses, 0xFFFFFF00 to 0xFFFFFF2C.

Example 2.14

Consider memory map for an exemplary card in which there are sections at the memory allocation map. Consider its description in a *locator* program. The sections for an exemplary embedded system, smart card memory map [Figure 2.25(c)] may be defined as follows.

```
1. SECTIONS
```

```
2. {/* Stack Top Location for 128 B RAM*/
3. _TopOfStack = 0x10100;
4. /* Bottom of Heap */
5. _BottomOfHeap = 0x10080;
6. text rom :
7. {/* Debit-credit card program instructions are at the text file
   named here*/
8. * (----.txt)
9. }
10. data ram :
11. {
12. /* Shadow Segment for 16 byte of Initialised Data at RAM for a copy
   from ROM from */
13. _DataStart = 0x10000;
14. /* Debit-credit card shadow segment data at the data file named
   here*/
15. * (----.data)
16. _DataEnd = 0x1000F;
17. }
18. /* Command for copy into the RAM */
19. > rom
20. bss :
21. {
22. /* Base Segment for 32 byte of Program Variables Data at
   RAM */
23. _bssStart = 0x10010;
24. /* Smart card base segment data at the base segment data
   file named here*/
25. * (----.bss)
26. _bssEnd = 0x1002F;
27. }
28. }
```

Example 2.15

Consider another memory map [Figure 2.25(d)] for another exemplary card. The locator specifies different map sections as follows:

SECTIONS

```
1. {/* Stack Top Location for 4 kB RAM*/
2. _TopOfStack = 0x11000;
3. /* Bottom of Heap */
```

```
4.  _BottomOfHeap = 0x10900;
5.  text rom :
6.  {/* Encrypting Java Card program instructions are at
    the text file named here*/
7.  a. (-----.txt)
8.  data ram :
9.  {
10. /* Shadow Segment for 256 bytes of Initialized Data at RAM for a copy
    from ROM from */
11. _DataStart = 0x10000;
12. /* The card shadow segment data at the data file named here*/
          a. (-----.data)
13. _DataEnd = 0x100FF;
14. }
```

```
    /* Command for copy into the RAM */
15. a. rom
16. bss :
17. {
18. /* Base Segment for 2 kB Program Variables Data at RAM */
19. _bssStart = 0x10100;
20. /* Java card base segment data at the base segment data file
    named here*/
```

```
21. a. (-----.bss)
22. _bssEnd = 0x108FF;
23. }
24. }
```

The memory map that includes the device IO addresses is designed after appropriate address allocations of the pointers, vectors, data sets and data structures. If the main memory is of Harvard architecture, the program memory map will be separate. For example, 8051 reads from the program memory by a separate set of instructions (input–output instructions).

2.8 PROCESSOR SELECTION

A hardware designer must take into account following processor-specific features:

1. A processor, which can operate at higher clock speed, processes more instructions per second.
2. A processor gives high computing performance when there exist (a) Pipeline(s) and superscalar architectures, (b) pre-fetch cache unit, caches, and register-files and MMU and (c) RISC architecture.
3. A processor with register-windows provides fast context switching in a multitasking system.
4. A power-efficient embedded system requires a processor that has programmable auto-shut down feature for its units and programmability for disabling use of caches when the processing need for a function

or instruction set is not constrained by limit on execution deadline. Processor uses Stop, Sleep and Wait instructions, and special cache design.

5. A processor that has a burst mode accesses external memories fast, reads fast and writes fast.
6. A processor with an atomic operation unit provides hardware solution to shared data problems when designing embedded software, else special programming skill and efforts are to be made when program uses shared variables and data buffers among multiple tasks.
7. When coding in assembly language or designing compiler or locator, data may store in big-endian mode in a system and the lower order bytes store at higher address: for example, in Motorola processors. Data may also store in little-endian mode in a system. Lower order bytes store at lower addresses and vice versa: for example, in Intel processors. A processor may also be *configure* at the initial program stage big-endian or little-endian storage of words: for example, the ARM processors.

The StrongArm family processors from *Intel* and TigerSHARC from *Analog Devices* have high power efficiency features.

The processor selection processes can be understood by considering four representative cases. Firstly a design-table similar to Table 2.7 is built. Then a processor having the required structural units and capable of giving the desired processor performance in system is chosen.

1. *Case 1:* Systems in which processor instruction cycle time ~ 1 μs and on-chip devices and memory can suffice. Examples are automatic chocolate vending machine, 56 kbps modem, robots, data acquisition systems like an ECG recorder or weather recorder or multipoint temperature and pressure recorder and real-time robotic controller.
2. *Case 2:* Systems in which processor instruction cycle time ~ 10 to 40 ns required, on-chip devices and memory do not suffice and medium processor performance is required. Examples are 2 Mbps router, image processing, voicedata acquisition, voice compression, video decompression, adaptive cruise control system with string stability and network gateway.
3. *Case 3:* Systems in which instruction cycle times of 5 to 10 ns required and high MIPS or MFLOPS performance is needed. Examples are multiport 100 Mbps network transceiver, fast 100 Mbps switches, routers, multichannel fast encryptions and decryptions systems.
4. *Case 4:* Systems in which instruction cycle time of even 1-ns does not suffice and multi-processor system is required along with use of the floating point and MAC units. Examples are voice processing, video processing, realtime audio or video processing and mobile phone systems.

Different systems require different processor features. A hardware designer takes these into view and selects an optimum performance-giving processor.

2.8.1 Microcontroller Selection

There are numerous versions of 8051. Additional devices and units are provided in these versions. A version and microcontroller is selected for embedded system design as per the application as well as its cost.

1. Embedded system in an automobile, for example, requires a CAN bus (Section 3.10.2). Then a version with CAN bus controller is selected.
2. An 8051 enhancement 8052 has an additional timer.
3. Philips P83C528 has I^2C serial bus (Section 3.10.1).
4. 8051 family member 83C152JA (and its sister JB, JC and JD microcontrollers) has two direct memory access (DMA) channels on-chip. (Section 4.8) The 80196KC has a PTS (Peripheral Transactions Server) that supports DMA functions. [Only single and bulk transfer modes are supported, not the burst transfer mode.] When a system requires direct transfer to memory from external systems, the DMA controller, improves the system performance by providing for a separate processing unit for the data transfers from and to the peripherals.

Table 2.7 Essential processor capabilities in four exemplary set of systems

Processor capability Required	*Case 1: Automatic Chocolate Vending Machine, Data Acquisition System, Real time Robotic Control*	*Case 2: Voice data acquisition, Voice-data Compression, Video Compression, Adaptive Cruise Control System with String Stability, Network Gateway*	*Case 3: Multi-port Network Transceiver, Fast Switches, Routers, Multi channel Fast Encryptions and decryptions*	*Case 4: Voice Processor, Video processing and Mobile Phone Systems*
Required processor	Microcontroller	Microprocessor	Multiprocessor System	Microprocessor +DSP based Multiprocessor System
Processor instruction cycle in μs (typical)	~ 0.5 to 1	0.01 – 0.04	0.0005 – 0.001	0.001 – 0.0005
Processor performance	Low suffices	Medium to high	High	Very High
Internal bus width in bits	8	32	32	64
CISC or RISC architecture	Any	RISC	RISC	RISC
Program counter and stack pointers	16	32	32	32
Stack at external or internal memory	External	External or internal	Internal	Internal
On-chip atomic operations unit	–	–	Yes[1]	–
Pipelined and super-scalar and pipelined architecture	No	Yes	Yes	Yes
Off-chip RAM in view of excessive RAM needs	No, on-chip suffices	Yes	Yes	Yes
On-chip register windows and files due to fast context switching needs	No	Yes	Yes	Yes
Interrupts handler internal in micro-controller or external to processor	Internal microcontroller	External	External	External

(*Contd*)

Processor capability Required	*Case 1: Automatic Chocolate Vending Machine, Data Acquisition System, Real time Robotic Control*	*Case 2: Voice data acquisition, Voice-data Compression, Video Compression, Adaptive Cruise Control System with String Stability, Network Gateway*	*Case 3: Multi-port Network Transceiver, Fast Switches, Routers, Multi channel Fast Encryptions and decryptions*	*Case 4: Voice Processor, Video processing and Mobile Phone Systems*
Instruction and data caches and MMU	No	Yes	Yes	Yes
On-chip memory flash or EPROM	Yes, on-chip suffices	No, on-chip does not suffice	No, on-chip does not suffice	No, on-chip does not suffice
External interrupts	1 to 16	1-2	128-256	16-32
Bit manipulation instructions	Used	Heavily used	Heavily used	Heavily used
Floating point processor	No	Yes	No	Yes
Streams of data requiring Harvard main memory architecture	No	Mostly Not necessary	May be Yes	Invariably Yes
DMA controller channels	No	No	Yes	May be Yes
Exemplary processor family	8051, 68HC11 or 12 or 16, 80196, PIC16F84	80x86, 80860, 80960	ARM7, Sunspare	ARM9, TMS family DSPs, PowerPC

[1]Needed when multiple ports and multichannel operations need data sharing.

Example 2.16 Case Study of a Real-time Robot Control System

1. A robotic system motor needs signalling at the rate above 50 to 100 ms. Hence there is enough time available for signalling and real-time control of multiple motors at the robot when we use a processor with instruction cycle time ~1 μs.
2. The processor speed need not be very high and performance needed is much below 1 MIPS. So no caches and advanced processing units like pipeline and superscalar processing are required.
3. A four-coil stepper motor needs only a 4-bit input and a DC motor needs a 1-bit pulse width modulated output. Therefore an 8-bit processor suffices.
4. Frequent accesses and bit manipulations at IO ports are needed. CISC architecture therefore suffices.
5. The program can fit in 4 kB or 8 kB of internal ROM on-chip. Stack sizes needed in the program are small so that can be stacked in an on-chip 256 or 512-byte RAM. A microcontroller is thus needed. No floating-point unit is needed.

Microcontrollers appropriate for the above case are 8051, 68HC11, 68HC12, 68HC16 or 80196. Microcontrollers 68HC12 and 68HC16 can be used due to availability of large number of ports. The 68HC12's instruction cycle and clock cycle time equals 0.125 μs. Number of ports equals 12 in 68HC12. Therefore, 6 or more degree of freedom robot with 6 or more motors can be driven directly through these ports. STOP and WAIT instructions in the processor save power when the robot is at rest!]

Example 2.17 Case Study of Voice Data Compression System

1. Voice signals are pulse-code modulated. The rate at which bits are generated is 64 kbps. A suitable algorithm can process the data compression of these bits with an instruction cycle time of ~ 0.01 to 0.04 μs (100 to 25 MHz) when the processor uses advanced processing units and caches.
2. Let us assume that the processor instruction cycle time is 0.02 μs (50 MHz). With a three-stage pipeline and two-line superscalar architecture, the highest performance will be 300 MIPS. [Refer to Example 2.11 for an understanding of the computations of MIPS]. It suffices for not only for voice but also for video compression.
3. Frequent accesses and complex instructions may not be needed.
4. The program cannot fit in 4 kB or 8 kB of internal ROM on-chip, and stack sizes needed in the program are big. Instead large ROM and RAM as well as caches are needed.
5. No floating-point is needed as mostly the bit manipulation instructions are processed during compression.

Exemplary processors that are appropriate for the above case are 80x86 and ARM family processors.

Example 2.18 Case Study of Fast Network Switching System

1. Transfer rates of 100 MHz plus are needed in fast switches on a network. Assuming 10 instructions per switching and transceiver action, instruction cycle time is ~ 0.001 plus. A multiprocessor system is needed for GHz transfer rates.
2. Let us assume that the processor instruction cycle time is 0.01 μs (100 MHz). With a five-stage pipeline and two-line superscalar architecture, the highest performance will be 1000 MIPS. [Example 2.11]. Multiprocessor system is thus needed for 1000 MHz plus switches.
3. The processor should have RISC architecture for single cycle instruction processing at each stage and line.
4. ROM and RAM as well as caches are required.
5. No floating-point is needed as mostly the bits are processed for IOs.

Exemplary processors that are appropriate for the above case are ARM7, ARM9 and Pentium.

Example 2.19 Real-Time Video Processing

1. Real-time video processing requires fast compression of an image needing use of DSP. Many real-time tasks have to be processed: for instance, scaling and rotation of images, corrections for shadow, colour and hue, image sharpening and filter functions. In such cases, a multiprocessor system with DSP(s) and that has the best processing performance is required.

Exemplary processors that are appropriate for multiprocessor system are ARM9 integrated with TMS family DSP(s) or ARM11 or TigerSHARC.

2.9 MEMORY SELECTION

Once the software designer's coding is over and the ROM image file is ready, the hardware designer is faced with the questions of what type of memory and what size of each should be used. First a design-table, as in Table 2.8, is built. The memory having the required features and address space is chosen. Following are the case studies. The actual memory requirement is known only after coding as per the design functions and specifications. ROM and RAM allocations for various segments, data sets and structures will be available from the software design. However, a prior estimate of the memory type and size requirements can be made. [Remember, the memory are available as: 1 kB, 4 kB, 16 kB, 32 kB, 64 kB, 128 kB, 256 kB, 512 kB and 1 MB. Therefore, when 92 kB of memory is needed, then a device of 128 kB is selected.]

Example 2.20

(a) Case Study of an Automatic Washing machine

Consider an automatic washing machine system. Assume that machine is not saving the pictures and graphics. (a) An EEPROM's first byte is required to store the state (wash, rinse cycle 1, rinse cycle 2 and drying) that has been completed. The second byte is required to store the time in minutes already spent at the current stage. The third byte is needed to store the status of the user set buttons. Thus a 128 B EEPROM at best should suffice in microcontroller. (b) Embedded software can be within 4 kB ROM at the microcontroller. (c) RAM is needed only for a few variables and stacks. An internal RAM of 128 B should suffice. (d) Therefore, no external memory is required with the system when using a microcontroller.

(b) Case Study of a Robotic

Consider a robotic system. (a) EEPROM bytes are required to store the rest status of each degree of freedom. Thus 512 B EEPROM in the microcontroller at best should suffice. (b) Embedded software can be within 32 kB ROM in the microcontroller. (c) RAM need is only for the variables and only one stack is needed for the return address of the subroutine calls. Internal RAM of 512 B should suffice. Therefore, no external memory is required with the system when using a microcontroller.

Example 2.21

(a) Case Study of the Data Acquisition Systems for the sixteen-parameter channels and voice/image processing during acquisition

Consider a data acquisition system. Assume that there are sixteen channels and at each channel 4 B of data store every minute. (a) Bytes are to be stored in flash memory. Assume that the results are stored in flash memory for a day before it is printed or transferred to a computer. Thus 92 kB is the data acquired per day. A 128 kB flash memory will thus suffice. (b) Embedded software can be within 8 kB ROM in the microcontroller. (c) RAM is needed only for the variables and only one stack is needed for the return address of the subroutine calls. An internal RAM of 512 B will suffice. (d) Intermediate calculations are needed for storing ADC results in the proper format. Unit conversion functions need to be calculated, which may necessitate a RAM of about 4 kB to 8 kB. (e) Therefore, a microcontroller with 8 kB EPROM and 512 B RAM is required, and an external flash (or 5 V EEPROM) of 128 kB and external RAM of 4 to 64 kB are required with the system. For acquiring image or voice data on-line, RAM buffer requirement can be 512 MB.

(b) Case Study of the Data Acquisition Systems for the ECG waveforms

Consider another data acquisition system, which is used for recording the ECG waveforms. Let each waveform be recorded at 256 points. A 64 kB flash will be required for 256 patient records.

Table 2.8 Required memory in four exemple of systems

Memory Required	*Case 1: Automatic Chocolate Vending Machine or Real time Robotic Control system*	*Case 2: Data Acquisition System*	*Case 3: Multi-port Network Transceiver, Fast Switches, Routers, or Multi channel Fast or Encryption and decryption system*	*Case 4: Voice Processor or Video processing or Mobile Phone or Pocket PC System*	*Case 5: Digital Camera or Voice Recorder System*
Processor used	Micro-controller	Micro-controller	Multiprocessor system	Microprocessor + DSP-based Multiprocessor system	Micro-processor
Internal ROM or EPROM	4 to 32 kB	8 kB	–	–	–
Internal EEPROM	256 to 512 B	256 to 512 B	–	–	–
Internal RAM	256 to 512 B	256 to 512 B	–	–	–
ROM or EPROM device	No	No	64 kB	64 MB	64 kB
EEPROM or Flash device[1]	No	64 to 128 kB	512 B	32 kB to 256 kB 2 to 8 GB memory stick	Flash 16 MB to 8 GB memory stick
RAM device	No	64 kB to 512 MB	64 kB to 512 MB	8 MB	1 MB
Parameterised distributed RAM	No	No	Yes for IO buffers. 4 kB per channel		–
Parameterised Block RAM	No	No		Yes for MAC unit, Dialing IO unit	–

Note: [1]Flash with a boot block can be used to store the protected part of the boot program in its OTP sector(s).

Example 2.22 Case Study of a multichannel Fast Encryption cum decryption Transceiver Systems

1. Consider a system with multiple channels. There are encrypted inputs at each channel. These are decrypted for retransmission to other systems.
2. EEPROM is required for configuring ports and storing their statuses. Assume 16 channels. 512 kB will suffice for a 16 B need per channel.
3. Encryption and decryption algorithms can be in 64 kB ROM.

4. Multichannel data buffers are required before the caches process the algorithms. Therefore, 1 MB to 512 MB RAM may be required.
5. IO buffer storage of 4 kB per channel is needed. If a parameterised distributed RAM is employed at each channel, the system performance will be increased.
6. The system will thus need the following memory: 64 kB ROM, 512 B EEPROM, 1 MB RAM and 4 kB per channel distributed parameterised RAM.

Example 2.23 Case Study of a Mobile Phone system

1. As voice compression–decompression and encryption–decryption algorithms and DSP processing algorithms are required, the ROM image will be large. Assume it is can be taken as 64 MB. Now if the ROM image is stored in a compressed format, a boot-up program first runs a decompression program. The decompressed program and data first load at RAM and then the application program runs from there. The RAM is obviously of bigger size in these systems. ROM can be reduced as per compression factor.
2. A large RAM is also needed. It can be taken as 8 MB for storing the decompressed program and data and for the data buffers.
3. The phone memory for entering important telephone numbers can be in a 16 kB flash or EEPROM. A flash of 64 kB can be taken for recording messages, MMS pictures. A memory stick using SDIO port of 2 GB or 8 GB is required for recording songs and videos.
4. Parameterised block RAM at MAC subunit and other subunits will improve system performance.
5. The system will thus require memory of 1 MB ROM, 16 kB EEPROM, 16 kB Flash, 1 MB RAM and block RAM at the subunits.

Example 2.24 Case Study of Digital Camera and voice recorder

1. Assume a low-resolution uncoloured digital camera system. Images are to be recorded GIF (graphic image format) compressed format. (a) Assume that an image has a Quarter-CIF (Common Intermediate Format) of 144 x 176 pixels. Then, 25,344 pixels are to be stored per image. Assume that compression reduces the image by a factor of 8 then 3 kB per image will be needed. Flash required will be 0.2 M for 64 camera images. Therefore, a 256 kB flash will be required. (b) A 64 B digital uncoloured images camera system will thus be estimated to need memory of 64 kB ROM, 256 kB flash and 1 MB RAM. High resolution 6 M pixel digital camera need 16 MB 256 MB flash and 2 to 8 GB memory stick.
2. Assume a voice recording system. 64 kbps are required, assuming an 8-bit pulse code modulation of the voice signals. [Average frequency is taken as 8 kHz.] Assume voice data compression factor of 8, 1 kB flash is required per second. A is MB flash 4 required for each hour of recording.
3. Since voice compression–decompression algorithms have to be processed, the ROM image will be large. It can be taken as 1 MB. Using compression techniques, a 64 kB ROM can store the ROM image.
4. The RAM needed is large for storing the decompressed program. It can therefore be estimated as 1 MB.
5. A one-hour voice recorder system is estimated to require memory or 64 kB ROM, 4 MB flash and 1 MB RAM.

§ *Simple systems like automatic chocolate vending machines or robots needs no external memory. The designer selects a microcontroller that has an on-chip memory required by the system. The data acquisition system needs EEPROM or flash. A mobile phone or pocket PC or digital camera system needs 1 MB plus RAM device and 64 kB to 256 kB internal flash device plus memory stick. Image or voice or video recording systems require a large flash memory in the form of memory stick of 2 GB to 8 GB.*

Summary

- 8051 microcontroller has Harvard architecture for memory, has special function registers, internal RAM and ROM or flash. It has two timers, and SI interface for the half duplex synchronous and full duplex UART communication.
- Bus signals interface the processor, memory and devices. The interface circuit takes into account the timing diagram with reference to processor clock output. The circuit uses the processor control, address and data bus signals and takes into account the timing diagram for the bus signals. A PAL-, GAL- or FPGA-based circuit, called glue circuit, provides a single-core or chip solution for the latches, decoders, multiplexers, demultiplexers and other necessary interfacing circuits.
- The buses interface to stepper motor, LCD controller, A/D, D/A circuits using ports and appropriate interface circuit.
- An embedded system hardware designer must select an appropriate processor, appropriate set of memories for the system and design an appropriate interfacing circuit between the processor, memories and IO devices. This is done after taking into account the various available processors, structural units and architecture, memory types, sizes and speeds, bus signals and timing diagrams.
- The structural units of a processor that interconnect through a bus are memory address and data registers, system and arithmetic unit registers, control unit, instruction decoder, instruction register and arithmetic and logical unit. *Registers in processor* also called register set(s) window(s) or file(s) are important and are meant for various functions such as context switch to process another task or ISR.
- Advanced processors have following additional structural units—prefetch control unit, instruction queuing unit, caches for instruction, data and branch transfer, floating point registers and floating point arithmetic unit. Pipelining and superscalar features and caches in the processors are used in high performance systems. MIPS or MFLOPS or Dhrystone per second define the computing performance. The goal is to provide optimal computing performance at the least cost and power dissipation.
- ARM, SHARC, TigerSHARC and DSPs processors are used in high performance computations.
- A system needs ROM and RAM memory of various types and address spaces. Various forms of ROM are masked ROM, PROM, EPROM, EEPROM, flash, boot-back flash and memory stick or card. Basic details of the memories are the addresses available, speed for read and write operations, and modes of memory access.
- The Processors and Memory selection can be done using appropriate design tables.
- Each IO device has a distinct set of addresses. Each IO device also has a distinct set of device registers – data registers, control registers and status registers. At a device address, there may be more than one register. The device addresses depend on the system hardware. Based on the memory map with IO device addresses, a locator program is designed to locate the linked object code file and generate a ROM image.

Keywords and their Definitions

Absolute addressing mode : Define all the address bits in an instruction.

Accelerator : ASIC, IP core or FPGA, which accelerates the code execution and which may also include the bus interface unit, DMA, read and write units and registers with their cores.

Accumulator : A register that provides input to an ALU and that accumulates a resulting operand from the ALU.

AHB : A high-performance version of the AMBA used in ARM processors.

ALU : A unit to perform arithmetic and logic operations as per the instructions.

AMBA : An established open source specification for on-chip interconnects that serves as a framework for SoC designs and IP library development.

Arithmetic unit registers : Registers that hold the input and output operands and flags with the ALU.

ARM : A family of high performance reduced code density ARM7, ARM9, ARM10 and ARM 11 processors, which are used in embedded systems as a chip, or as a core in an ASIC.

ARM7 and ARM9 : Two families of RISC processor for an SoC from ARM and Texas Instruments, also available in single-chip CPU versions and in file versions for embedding at a VLSI chip. ARM 7 has Princeton architecture for the main memory and ARM9 has Harvard Architecture.

Asynchronous serial : Data bytes or frames not maintain uniform phase differences in serial

communication : communication.

Auto index : When after executing an instruction, the index register contents change automatically.

Base addressing : Addressing an address from where a first element of data structure starts.

Baud Rate : Rate at which serial bits are received at the line during a UART communication.

Boot back flash : A flash with a few sectors similar to an OTP device, to enable storage of bootup program and data.

Branch transfer cache : Cache to hold in advance the next set of instructions to be executed on the program branching to this set.

Burning-in : A process in which bits are modified from all 1s in erase form to the 1s and 0s in a device as per input 1s and 0s.

Bus interface unit : A unit to interconnect the internal buses with the external buses for control, address and data bits.

CISC : A complicated Instruction Set Computer that has one feature that provides a big instruction set for permitting multiple addressing modes for the source and destination operands in an instruction. The hardware executes the instructions in a different number of cycles, as per the addressing mode used in an instruction.

Code Compatibility : Usability of codes by various generations of a family.

Code composer studio : An IDE for TI DSP-specific code composing which provides an environment similar to MS Visual C++. It consists of the following: multilevel (C as well as DSP assembly) debugger, compiler, assembly optimiser, RISC-like assembly codes and RISC-like scheduling for optimum performance and efficiency probe points, file IO functions, comprehensive data visualization displays and GEL scripting language based on C.

CODEC : A unit for digital coding after ADC and other operations and decoding to get analog signals using DAC and other operations. It is used in processing audio or CCD device pixels and video signals.

Control unit : To control and sequence all the processing actions during an instruction execution.

DAA : Direct access arrangement; for example, a typical DAA serial in and out port directly transferring the analog input and output using up to 1 master and 7 slave CODECs.

Data cache : Cache to hold the data in content-addressable memory format.

DCT : Discrete Cosine Tranformation function used in a number of DSP functions, for example, the MPEG2/MPEG4 compression.

Device address : A device address used by processor to access its set of registers. At each address there may be one or more device registers.

Device programmer : A system or unit for programming a device by burning-in ROM image.

Device programming : Programming of bits by burning-in a memory of microcontroller or in a PLA, PAL, CPLD or any other device.

Device register : A register in a device for byte, word of data, flags or control bits. Several device registers may have a common address.

Device : A physical or virtual unit that has three sets of registers: data registers, control registers and status registers, and which the processor addresses it like a memory.

Dhrystone : A benchmarking program that measures the performance of a processor for processing integers and strings (characters). It uses a benchmark program available in C, Pascal or Java. It benchmarks a CPU, not the performance of the IO or OS calls. 1 MIPS = 1757 Dhrystone/s.

Digital Filtering : A filter for the signals that use DSP functions.

Direct address : A directly usable address in an instruction. It is usually the address on a page in the memory.

DMA : A direct memory access by a controller internal or external. DMA operations facilitate the peripherals and devices of the system to obtain access to the system memories directly, without the processor controlling the transfer of the bytes in a memory block.

DRAM : Dynamic RAM, which refreshes continuously by a device called DRAM refresh controller. Once programmed, it auto reads and writes the same set of bits repeatedly by scanning the DRAM memory cells.

Echo cancellation : A process of eliminating echoes.

Echo : A signal received after a delay and which superimposes over the original signal. For example, in a hall or at the hills we hear the original sound as well as the echoed sound. Similarly, there may be echo in electronic signal.

EDA : A powerful tool for Electronic Design Automation.

EEMBC : EDN Embedded Benchmark Consortium.

EEPROM : A type of memory each byte of which is erasable many times and then programmable by the instructions of a program as well as by a device programmer.

EPROM : A type of memory that is erasable many times by UV light exposure and programmable by a device programmer.

Erase time : Time taken for device erasing.

Fixed point arithmetic : Arithmetic using signed or unsigned integers employing processor registers or memory.

Flash : A memory in which a set of sectors erase simulatenously.

Flash : A type of memory in a sector of bytes that is erasable many times (maximum, ~10000) in a flash at the same instance in a single cycle. Each erased byte is then programmable by the write instruction of a program as well as by a device programmer.

Floating point arithmetic : Arithmetic using processor registers or memory, where the decimal numbers and fractional numbers are stored in a standard floating-point representation.

High-level language support : A supporting unit for given processor structure that facilitates program coding in C or other high level languages and enables their running like machine codes by an internal compilation.

Index register : A register holding a memory address of a variable in an array, queue, table or list.

Instruction cache : A cache to sequentially hold the instructions that have been prefetched for pipeline base parallel execution.

Instruction decoder : The circuit to decode the opcode of the instruction and direct the control unit accordingly.

Instruction format : Format of expressing an instruction.

Instruction queuing unit : A unit to hold a queue of instructions and place these into the cache.

Instruction register : A register to hold the current instruction for execution.

Instruction set : A definite set of executable instructions in a processor.

Instruction set : A unique processor-specific set of instructions.

Interface circuit : A circuit consisting of the latches, decoders, multiplexers and demultiplexers.

Internal bus : A set of paths that carry in parallel the signals between various internal structural units of a processor. Its size is 64-bit in a 64-bit processor.

Java Accelerator : An accelerator that helps in the execution of Java codes faster than a JVM.

JVM : Machine codes that use the compiled byte codes of a Java program and run the program on a given system.

MAC unit : A unit used in DSP operations for fast calcuation of $\Sigma[ax_n + (b_i.y_j)]$ or similar expressions.

Master : A processor, device or system which synchronously or asynchronously controls the output to several different processors, devices and systems called slaves.