Maurizio Di Paolo Emilio

Embedded Systems Design for High-Speed Data Acquisition and Control

Springer

Embedded Systems Design for High-Speed Data Acquisition and Control

Maurizio Di Paolo Emilio

Embedded Systems Design for High-Speed Data Acquisition and Control

Springer

Maurizio Di Paolo Emilio
Pescara
Italy

ISBN 978-3-319-06864-0 ISBN 978-3-319-06865-7 (eBook)
DOI 10.1007/978-3-319-06865-7

Library of Congress Control Number: 2014946192

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Imagination is more important than knowledge.

A. Einstein

When wireless is perfectly applied, the whole earth will be converted into a huge brain, capable of response in every one of its parts.

N. Tesla

To Julia, Elisa and Federico

Forewords

It's a great pleasure for me to write the Preface of a book which aims to help to implement one of the oldest technologies (data acquisition) using one of the concepts (embedded system) that will reshape the future of the world. The first known acquisition system used was by Egyptian pharaohs in 2500 BC to record the water flow of the Nile to predict flood. From this ancestor engraving down values on stone, with an imaginary time travel, we arrive in 1963: in that year was invented the first data acquisition systems. In the past, data acquisition systems were largely mechanical, using smoked drums or chart recorders. Today, powerful microprocessor performs data acquisition faster, more accurately, more flexibly, with more sensors, more complex data processing and elaborate presentation of the final information. Data acquisition systems can now be used as application demand, wired or wireless. Mobile devices is another area which has grown in adoption and most of them have many kinds of data acquisition systems built in. Now the data acquisition industry may be at the beginning of another revolutionary data acquisition change with the advancements of the cloud computing on the Internet. Benefits of cloud computing include real-time access of information, scalability of processing power, less risk of downtime and direct interaction with other web services. The embedded systems industry, on the other hand, is very recent: it was born with the invention of microcontrollers and since then it has evolved into various forms, from primarily being designed for machine control applications to various other new verticals with the convergence of communications. The importance of embedded systems is growing continuously. Exponentially increasing computing power (Moore's law), ubiquitous connectivity and convergence of technology have resulted in hardware/software systems being embedded within everyday products and places. In fact, 98 % of all computing devices manufactured today go into embedded systems. By definition (I like the definition given by Edward Lee—Berkeley—of Cps systems, integration of computation with physical processes) embedded systems are an optimized mix of hardware and software. So another key word is codesign. In synthesis, the reach of system-level objectives by exploiting the trade-offs between hardware and software in a system through concurrent design. Embedded developers must have deep insight into both the

hardware and software sides of the design. This is the goal, and the usefulness, of Maurizio's book: provide a solid groundwork for all embedded developers—and for all the people approaching this exciting world—involved in the "codesign" of data acquisition systems in sync with the existing and future technology, in both hardware and software.

Milan, June 2014 Filippo Fossati

Technology marches on, almost relentlessly, as every aspect of our lives demands newer, smaller, more feature-rich and comfort-and convenience-driven electronics at an unabated pace. At the heart of all these devices, however small, are embedded systems. It has been said that one of the very first recognizably modern embedded systems was the Apollo Guidance Computer, developed by the MIT's Labs. But, however technically impressive, this guidance computer might have been used and its use of the then newly developed integrated circuits (ICs) to reduce the system's size and weight had sent shivers down many a space scientists' spines not to mention those footing the bill for the mission; the guidance computer was considered the riskiest item in the Apollo project. This granddaddy of embedded systems did humankind proud: not only did it go to space, but also it duly verified the use of ICs into electronics systems, however mission critical they may be. Since then, the use of embedded systems has skyrocketed too. Such systems are found in just about every electronics system we can think of. But, to keep embedded systems in tune with all the other developments that go hand in hand, we—the editors, authors, engineers, lecturers, etc.—should continue keeping all involved parties up-to-date. Maurizio is an author who has dedicated his efforts to learning and working with embedded systems, as well as data acquisition and control systems, and has translated that knowledge and experience into information and education for the practicing and learning engineers alike. His work can be regularly read about on the pages of Electronics World magazine to which he has been a regular technical contributor—with his own column—for many years. Topics he regularly covers in the magazine range from basic electronics through data acquisition and control systems to embedded systems design. As embedded systems permeate every sphere of our lives, nowadays they are also found in data acquisitions systems. Data acquisition has its own historical record too—its early beginnings can be traced back to 1963, when IBM produced computers that specialized in data acquisition. Among the earliest models were the IBM 7700 and IBM 1800 Data Acquisition and Control Systems, which were rather expensive. Later, these systems were superseded by the general-purpose computer and specialist data acquisitions cards. Today data acquisition systems are much smaller and less expensive than their first versions, and also rely on embedded systems for their successful operation. As such, Maurizio is well placed to prepare this book and present his knowledge to an

even wider audience. So, electronics can continue on its inventive path, launching new technologies and rather revolutionary systems and devices at a high speed, as long as we have dedicated individuals such as Maurizio teaching us about them, we are in good hands to learn more and create more. Electronics is a fabulously creative discipline, and all of us involved in this field feel truly blessed to be a part of it.

London, June 2014 Svetlana Josifovska

Preface

Embedded systems are a combination of hardware and software, which facilitates mass production and a host of applications, benefiting from economies of scale. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. Some applications of embedded system are in the automotive filed as control system and in the data acquisition for scientific and industrial fields (control system).

Modern industrial plants utilize robots for obtaining temperature controls, pressure controls, speed controls and position controls, and so on. Feedback control system is to be found in almost every aspect of our daily environment.

Data Acquisition Systems (DAQ) are now the main instruments for testing, measuring in automation field and so on, which is used widely in the research laboratory by scientists and engineers.

Data acquisition is a necessity, which is why data acquisition systems and software applications are essential tools in a variety of fields. For instance, research scientists rely on data acquisition tools for testing and measuring their laboratory-based projects. Therefore, as a data acquisition system designer, you must have in-depth understanding of each part of the systems and programmes you create.

The complexity of new physics experiments and industrial processes require more complex DAQ with the following characteristics:

- Capable of managing large amounts of data.
- High-speed connection.
- Digital recording.
- Full reconfigure possibility.

Signals that are hard to characterize and analyse with real-time display are evaluated in terms of the following parameters:

- High frequency.
- Large dynamic range.
- Gradual changes.

Data acquisition software is typically available in a text-based user interface (TUI) that comprises an ASCII configuration file and a graphic user interface (GUI), which are generally available with any web browser. Both interfaces enable data acquisition system management and customization, do not need to recompile the sources. This means even inexperienced programmers can have full acquisition control.

Well-designed data acquisition and control software should be able to quickly recover from instrumentation failures and power outages without losing any data. Data acquisition software must provide a high-level language for algorithm design. Moreover, it requires data archiving capability for verifying data integrity.

You have many data acquisition software options. An example is programmable software that uses a language such as C. Other software and data acquisition software packages enable you to design the custom instrumentation suited for specific applications (e.g., National Instruments's LabVIEW and MathWorks's MATLAB).

A data acquisition system's complexity tends to increase with the number of physical properties it must measure. Resolution and accuracy requirements also affect a system's complexity. To eliminate cabling and provide for more modularity, you can combine data acquisition capabilities and signal conditioning in one device.

Recent developments in the field of fiber-optic communications have shown longer data acquisition transmission distances can cause errors. Electrical isolation is also an important topic. The goal is to eliminate ground loops (common problems with single-ended measurements) in terms of accuracy and protection from voltage spikes.

Recently, some new technological developments have proven to be beneficial to the overall efficacy of data acquisition applications. For instance, in USB flash drive successfully makes the data acquisition and storage simpler and more efficient than ever (think "plug and play") and wireless improves the speed of data transmission and security.

In future, consumers' demand for mobile computing systems will only increase, and this will require tablet computers to feature improved data acquisition and storage capabilities. Having the ability to transmit, receive and store larger amounts of data with tablets will become increasingly important to consumers as time goes on. There are three main things to consider when creating a data acquisition-related application for a tablet. Hardware connectivity: Tablets have few control options (e.g., Wi-Fi and Bluetooth). Program language support: Many tablets support Android apps created in Java. Device driver availability: Device drivers permit a high-level mode to easily and reliably execute a data acquisition board's functionality. C and LabVIEW are not supported by Android or Apple's iOS. USB, a common DAQ bus, is available in a set of tablets. In the other case, an adapter is required. Among these examples, moving a possible data acquisition system to a tablet requires extra attention.

For all of the aforementioned reasons that embedded system will figure prominently in the evolution of acquisition system technology makes them ideal for custom data acquisition systems and control system.

The limited function required of embedded systems allows them to be designed for the most efficient performance.

Such embedded computing and information technologies have become, at the same time, an enabler for future manufacturing enterprises as well as a transformer of organizations and markets.

Digital embedded security is no more an option but a necessity as it is critical for more transactions happening over embedded devices as front ends. Due to constrained resources on systems, embedded systems have challenges in implementation on full-fledged security systems; therefore, the concept of "embedded security" offers a new differentiator for embedded product marketing.

The main idea of this book is to describe the theory of the embedded system with the realization of a versatile project (hardware and software) for application as high-speed data acquisition and programmable control system.

Starting from the review of analogue and digital electronics, the book aims to provide the reader into a competent and independent practitioner in the field of embedded systems by providing several skills, in both hardware and software development.

On the hardware side, the book will focus on among microcontroller design, techniques of embedded design, high-speed data acquisition (DAQ) and control system. This culminates in the study and application of a Real-Time Operating System (Open Source), representing the most important way that an embedded system can be programmed. It is presented as a useful tool for embedded designers. Every concept has been made to present the many complex concepts in a way that is easy to understand and which makes them readily usable. Embedded Linux (both the free and licensed versions) remains an attractive choice for a range of development teams and its use is poised to see a manifold increase.

In the chapter of microcontroller design, techniques of design of FPGA will also be presented. FPGA designs combine multiple components into a single package that reduces component count, board size and manufacturing complexity. Processors, memory, custom logic and many of the peripherals in a typical embedded project can be found in the FPGA. Today's FPGA architecture has grown into billions of logic blocks (equivalent to gates), and with programmable interconnection flexibility designers can easily create hardware functions that exactly match the needs of a specific embedded application.

Moreover, embedded development system and PCB techniques will be presented. An embedded system is identified as the electronic device designed for a particular function.

The design of embedded system makes use of compilers, assembler, debugger and a whole range of suites for the development of both software and hardware.

PCB layout is one of the last steps but the most critical in the design process. High-speed circuit performance is heavily dependent on layout. A high-performance design can be rendered useless due to a poor or sloppy layout.

For a long time, embedded devices were mostly operating as stand-alone systems. However, with the advent of wireless connectivity, like Bluetooth, Zigbee, RFID, the scenario has changed. The recent trends in wireless for use in embedded systems are in the areas of system-on-chip (SoC) architecture, reduced power consumption and application of short-range protocols.

In future, security in the embedded devices will be a critical issue. The security requirements of the connected embedded devices are distinct according to their limited memory, constrained middleware and low computing power. Today, power consumption is still a key issue in the design of the embedded systems that directly affects the battery life, which the technology has not been able to match the advancements in the hardware that drives these systems in recent years.

Pescara, Italy, June 2014 Maurizio Di Paolo Emilio

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this book. In particular, I would like to thank Charles B. Glaser Executive Editor, Applied Sciences USA, for the publication of the present book. Many thanks to Filippo Fossati, Editor-in-Chief of Fiera Milano Media Electronic Division and Svetlana Josivofska, Editor of Electronics World, for the contribution of the Foreword. I want to thank my family for their patience, and also for encouraging and inspiring me to follow my dreams. I am especially grateful to my wife, Julia, for her support in terms of linguistic form and also to Albert Einstein and Nikolas Tesla, because without them today's innovative world would not exist.

Contents

Chapter 1
Review of Microelectronics

Abstract With the developing in the field of electronics is become more useful in the embedded system in particular in the Data Acquisition System (DAQ). The goal of this chapter is to review briefly the main features of electronics and microelectronics; one salient feature of this review is its synthesis and design-oriented approach.

1.1 Introduction

The earliest electronic circuits were fairly simple. They were composed of a few tubes, transformers, resistors, capacitors, and wiring. As more was learned by designers, they began to increase both the size and complexity of circuits. Component limitations were soon identified as this technology developed. The transition from vacuum tubes to solid-state devices took place rapidly. As new types of transistors and diodes were created, they were adapted to circuits. The reductions in size, weight, and power use were impressive.

Microelectronic technology today includes thin film, thick film, hybrid, and integrated circuits and combinations of these. Such circuits are applied in digital, switching and linear (analog) circuits. Because of the current trend of producing a number of circuits on a single chip, you may look for further increases in the packaging density of electronic circuits. At the same time you may expect a reduction in the size, weight, and number of connections in individual systems. Improvements in reliability and system capability are also to be expected [1, 2].

1.2 Basic of Semiconductor's Physics

Microelectronics is based on physics of structures. The main semiconductor elements used in the applications are silicon, boron, aluminum etc. Silicon atom has 4 valence electrons and its important for electrical system. At temperature close to 0 K, silicon crystal behaves as an insulator: its electrons are confined to their respectively covalent bonds. At higher temperature, electrons gain thermal energy and can be used as

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_1

"free charge". Bandgap energy, minimum energy to broke the bond, of the silicon is 1.12 eV where $1\,\text{eV} = 1.6 * 10^{-19}$ J. The number of "free electrons" that is possible to have in a semiconductor crystal depends on band gap energy and temperature, it's possible to define density of electrons (number of electrons per unit volume) as the following [3, 4]:

$$n = 5.2 * 10^{15} T^{\frac{3}{2}} * e^{-\frac{E_g}{2KT}} \tag{1.1}$$

with k = boltzmann constant = 1.38 $*10^{-23}$ J/K. For insulator materials $E_g \simeq 2.5$ eV, instead, for conductor is about less than 1 eV. As example we can calculate the density of electrons for silicon at T = 300 K:

$$n = 5.2 * 10^{15} T^{\frac{3}{2}} * e^{-\frac{1.72*10^{-19}}{2KT}} \sim 10^{10}\,\text{e/cm}^3 \tag{1.2}$$

The intrinsic semiconductors can't be used for electrical application due to very high resistance but also a small number of "free electrons" respect to the conductor. In an extrinsic semiconductor the situation is different: there is a p-Si and a n-Si. In the first case majority carriers (holes) are $p = N_A$, semiconductor is doped with a density of N_A (i.e. boron atom) and minority carriers $n = p_i^2/N_A$; in the other case, instead, majority carriers (electrons) $n = N_A$ with semiconductor is doped with a density of N_D (i.e. Phosphorus), minority carriers $p = n_i^2/N_D$. The concept of doping extends the flow of current in a semiconductor.

Drift

The drift is the movement of electrons charge due to an electric field: $v = \mu E$, μ is "mobility" expressed in $\text{cm}^2/\text{V} \cdot \text{s}$. If consider a piece of n-silicon ($\mu_n = 1{,}350$ $\text{cm}^2/\text{V} \cdot \text{s}$) of L = 1 μm applied a voltage of V = 1 V, the velocity of electrons (drift) is about $v = \mu_n E = \mu_n \frac{V}{L} \simeq 10^7$ cm/s. For a p-silicon $\mu_p = 480\,\text{cm}^2/\text{s}$ [1, 2].

Now, we should calculate the current due to this drift v (Fig. 1.1):

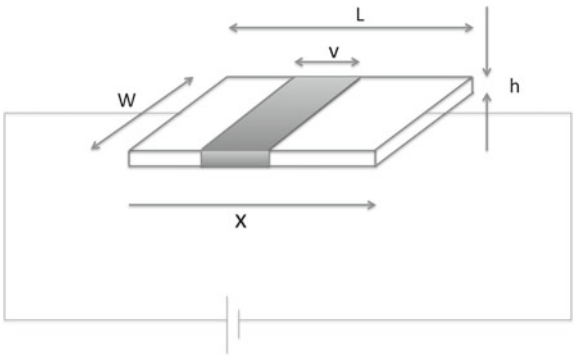


Fig. 1.1 Current flow in terms of charge density

$$I = -v \cdot W \cdot h \cdot n \cdot q \tag{1.3}$$

It can be written in another way:

$$J_n = \mu_n \cdot E \cdot n \cdot q \tag{1.4}$$

where J is the current density, $W \cdot h$ is the cross section and $n \cdot q$ is the charge density in Column of the semiconductor. In presence of both electrons and holes:

$$J = J_n + J_p = q(\mu_n n + \mu_p p)E \tag{1.5}$$

Diffusion
The mechanism of the diffusion is the movement of the charge from a zone of high concentration to zone of low concentration (current). The mathematica relation that explains this phenomena is the following:

$$I = A \cdot q \cdot D \cdot n \frac{dn}{dx} \tag{1.6}$$

where D_n is the "diffusion constant" in cm^2/s; in intrinsic silicon $D_n = 34\,\mathrm{cm}^2/\mathrm{s}$ (for electrons) and $D_p = 12\,\mathrm{cm}^2/\mathrm{s}$ (for holes). A is the cross section, q is the charge and $\frac{dn}{dx}$ is the concentration (electrons) respect to the direction "x" (Fig. 1.1). From the last equation we can calculate the current density for electrons and holes:

$$J_n = q D_n \frac{dn}{dx} \tag{1.7}$$

$$J_p = -q D_p \frac{dp}{dx} \tag{1.8}$$

1.2.1 PN Junction

From the doping we have obtained p-type and n-type semiconductors. A electric field or a concentration gradient leads to the movement of the charges (electrons and holes). We suppose to dope two adjacent pieces of semiconductors (Fig. 1.2); in this configuration there will be a flow of electrons from n to p side and a flow of holes in opposite direction (Fig. 1.3). At the end of this process of "equilibrium", a electric field (depletion region) will emerge as indicated in Fig. 1.3. The Junction reaches equilibrium once the electric field is strong enough to completely stop the diffusion current. The existence of electric field due to depletion region suggests that PN junction has a built-in potential defined in the following equation:

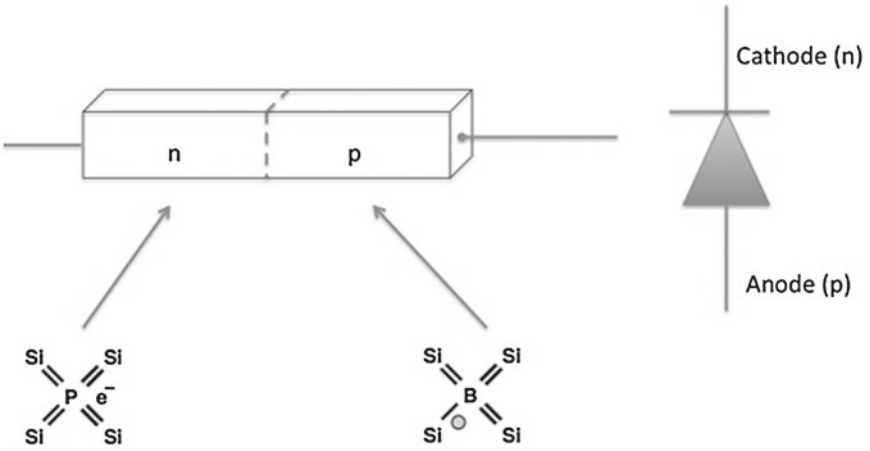


Fig. 1.2 PN junction

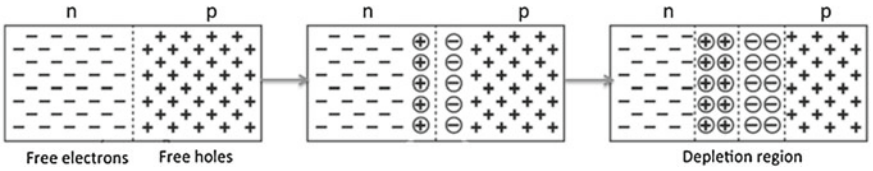


Fig. 1.3 Evolution of charge concentration in a PN junction

$$V(x_2) - V(x_1) = -\frac{D_p}{\mu_p} ln(\frac{p_p}{p_n}) \tag{1.9}$$

Considering the Einstein's equation, $\frac{D}{\mu} = \frac{KT}{q}$:

$$|V_0| = \frac{KT}{q} ln(\frac{p_p}{p_n}) \tag{1.10}$$

where p_n and p_p are the concentrations at x_1 and x_2 respectively. The PN junction so realized is called Diode. Having analyzed the PN junction in equilibrium, let us observe how it works applying an external voltage (Fig. 1.4):

Reverse Bias

The external voltage rises the electric field of depletion region prohibiting the flow of current. The device works as capacitor:

$$C_j = \frac{C_{j_0}}{\sqrt{1 - \frac{V_R}{V_0}}}; C_{j_0} = \sqrt{\frac{q\varepsilon_{s_i} q}{2} \frac{N_A N_D}{N_A + N_D} \frac{1}{V_0}} \tag{1.11}$$

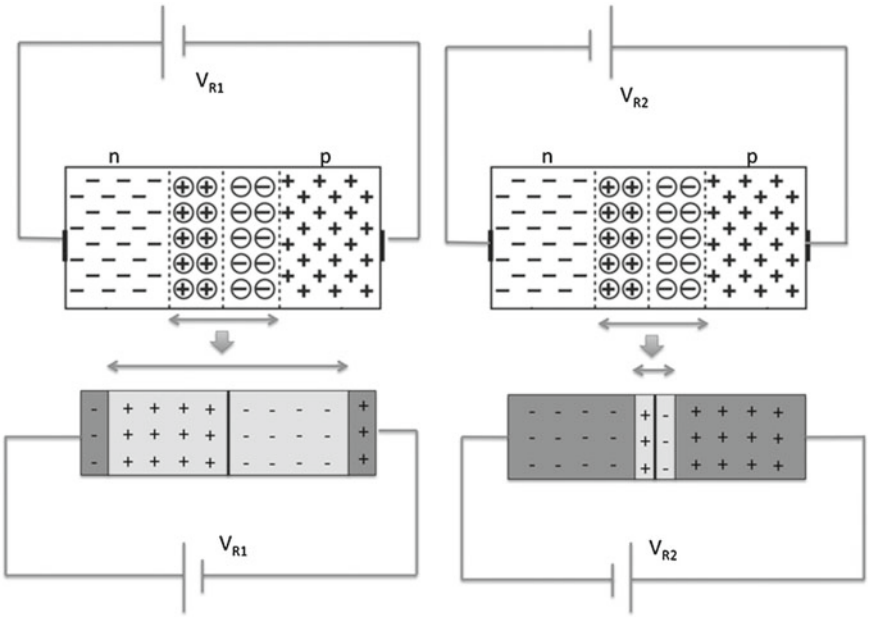


Fig. 1.4 PN junction with external voltage

ε_{s_i} is the dielectric constant of silicon.

Forward Bias
The external voltage decreases the electric field of the depletion area allowing greater diffusion current (Fig. 1.4):

$$I_D = I_s \left(exp\left(\frac{V_R}{V_T}\right) - 1 \right) \tag{1.12}$$

with V_T, thermal voltage and I_s:

$$I_s = Aqn_i^2 \left(\frac{D_n}{N_A L_n} + \frac{D_p}{N_D L_p} \right) \tag{1.13}$$

is the "reverse saturation current" and L_n and L_p are the electrons and holes "diffusion lengths" (i.e. $L_n = 20\,\mu\text{m}$, $L_p = 30\,\mu\text{m}$)

1.3 Diode

The diode is a two terminal device with I–V characteristic indicated in Fig. 1.5. Some application can be described in the following text [1]:

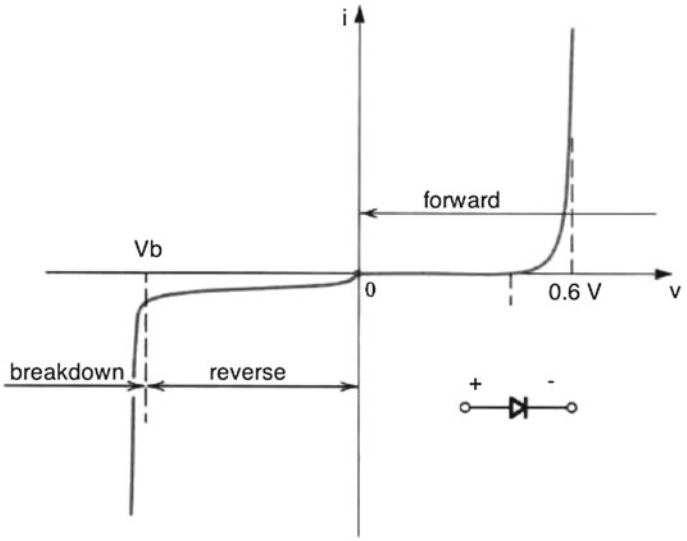


Fig. 1.5 Characteristic I–V of the diode

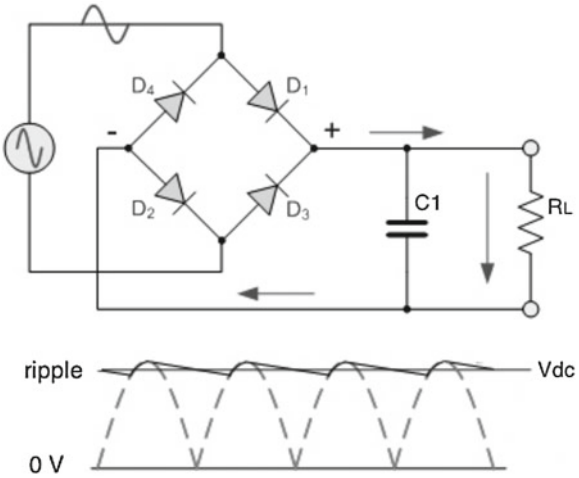


Fig. 1.6 Wave rectfier

- Wave rectifier: The circuit is visualized in Fig. 1.6: the ripple amplitude can be calculated by:

$$V_r \sim \frac{V_p - V_{D,on}}{R_L C_1 f_{in}} \tag{1.14}$$

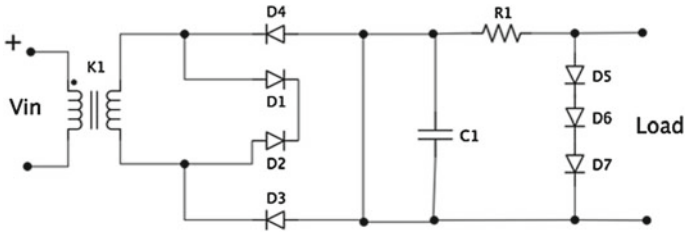


Fig. 1.7 Block diagram of voltage regulator with diodes

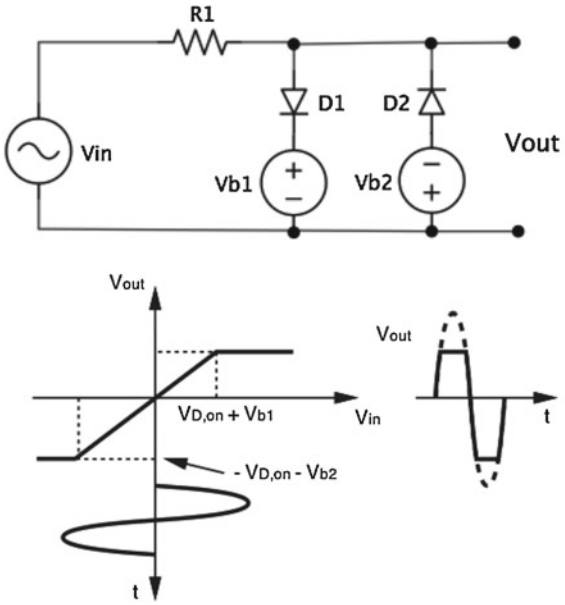


Fig. 1.8 Limiting circuit

- Voltage regulation: Due to significant variation of the line voltage, it is necessary produced in output of the diode a stabilized voltage. A possible outline is visualized in Fig. 1.7.
- Limiting circuit the circuit passes the input to the output, $V_{out} = V_{in}$ and when the input exceeds a "threshold" the output remain constant (Fig. 1.8).

1.4 Bipolar Transistor: Emitter Follower

The union of two junctions p-n (i.e. two diodes together) form the bipolar transistor junction (BJT). Bipolar because the current is sustained by electrons and holes

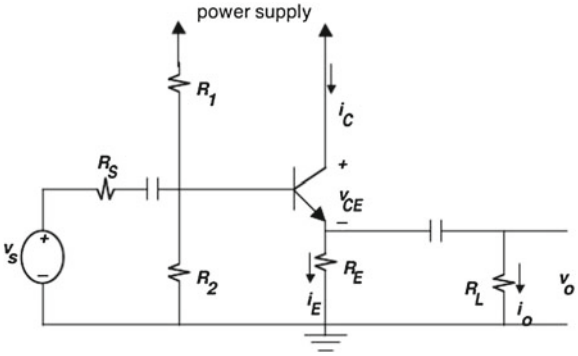


Fig. 1.9 Emitter follower (outline)

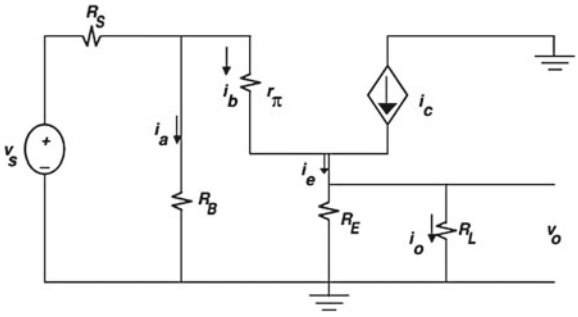


Fig. 1.10 Emitter follower (small signal equivalent circuit)

(such as the diode). Compared to the diode, the BJT (3 terminals) can be used as a signal amplifier. Although the MOS technology (see next paragraph) is more widespread, the technology bipolar remains significant (or predominant, in certain cases) in several applications:

- Electronics vehicle
- Systems wireless
- Digital Circuits ECL
- Draft discrete circuits

The emitter follower circuit is particularly useful for applications where high input impedance is required. It is typically used as a buffer in a wide variety of areas. Emitter follower is a common collector transistor configuration. It can be easily designed by circuit RC. The emitter follower is also known as a voltage follower, or a negative current feedback circuit, with high input impedance and low output impedance. The outline of the emitter follower is shown in Fig. 1.9 and the corresponding equivalent small signal circuit in Fig. 1.10. We can calculate the effective resistance seen from terminal B by:

$$R_{ib} = r_\pi + (\beta + 1)R_E//R_L \simeq (\beta + 1)R_E//R_L \tag{1.15}$$

This value is relatively high. In general, $R_E//R_L$ is around kΩ and $\beta \sim 100$, so the resistance seen from terminal B is in the hundreds of kΩ. It is evident from Fig. 1.10 that input resistance depends on load resistance. Let us named the input resistance as R_{in}:

$$R_{in} = R_B//R_{ib} \tag{1.16}$$

The effective resistance is the parallel between R_{ib} and R_b. A common-collector configuration can be used as an amplifier in such a circuit, where a large input resistance is needed; a good application can be a pre-amplifier circuit. From the voltage gain equation ($A = V_0/V_i$) follows:

$$A = \frac{V_0}{V_s} = \frac{R_E//R_L}{r_e + (R_E//R_L)} * \frac{R_{in}}{R_s + R_{in}} \tag{1.17}$$

It becomes evident that as long as $r_e \ll (R_E//R_L)$ and $R_s \ll R_{in}$ the gain approaches unity.

The emitter follower can be designed using the main steps described below:

- Choose a transistor: the transistor should be selected according to the system requirements.
- Select emitter resistor: selecting a working point (for example select an emitter voltage of about half the supply voltage).
- Determine the base current: base current is the collector current divided by β (or h_{fe}).
- Determine the base resistor values: select the value of the resistor(s) to provide the voltage required at the base.
- Determine the value of the input and output capacitor: The value of the input/output capacitor should equal to the resistance of the input/output circuit at the lowest frequency of operation.

When using the emitter follower circuit, there are two main practical points to note:

- The collector may need decoupling: in some cases the emitter follower may oscillate, in particular if long leads are present. One of the easier ways to prevent this is to decouple the collector to ground with very short connections, or by placing a small resistance between the collector and the power supply line.
- The input capacitance affects the RF: the base emitter capacitance may reduce the high impedance of the input circuit if the signal is above 100 kHz.

One of the designs of an emitter follower is shown in Fig. 1.11, with a P-spice simulation in Figs. 1.12 and 1.13. This configuration can be typically used in AC coupling applications. It's important that the frequency response and bias voltage on the base are planned for. In the circuit mentioned here, there's an added voltage divider, consisting of R_2 and R_3, and an AC coupling capacitor, C_1. Their component values will need to be calculated. An important aspect of the emitter follower is

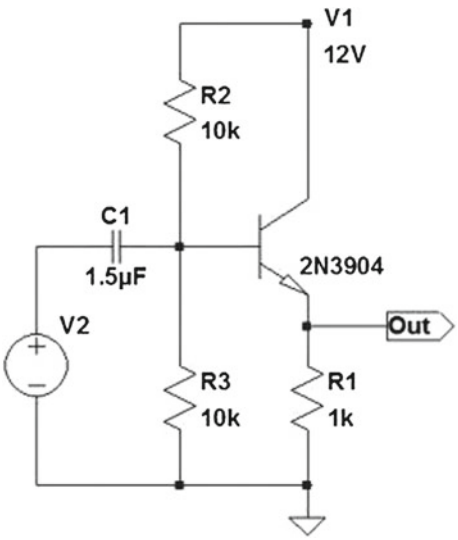


Fig. 1.11 Emitter follower (example with 2N3904)

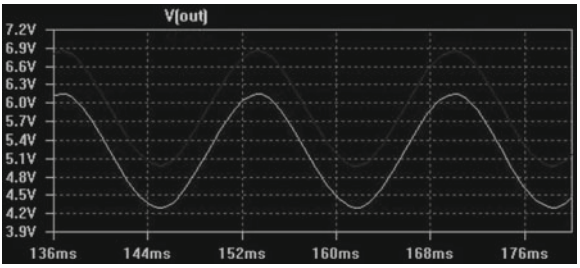


Fig. 1.12 Simulation of emitter follower, output voltage

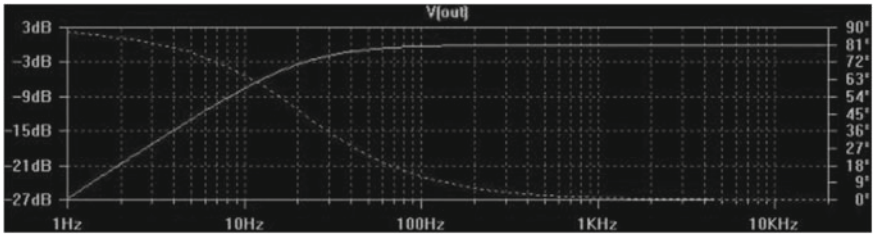


Fig. 1.13 Simulation of emitter follower, frequency response

its relative immunity to temperature instability. When the current in the collector increases (changing of β), the voltage across the emitter resistor also increases. This acts as a negative feedback, since it reduces the voltage difference between base and

emitter (V_{BE}), which drives the transistor to conduct more current, as such thermal stability is maintained.

1.5 MOS Transistor

Today the microelectronics is dominated by MOSFET devices. The physical structure of the MOSFET is described in Fig. 1.14: on a substrate of monocrystalline p-type, two junctions are made of type n+ which are connected two terminals called drain and source (D and S in the Fig. 1.14). In the area between the drain and source is made to grow a layer of silicon dioxide (thickness less than 0.01 μm), and that an excellent insulator [3–5].

The gate terminal is isolated from the Si substrate by a thin layer of SiO_2, and therefore the gate current DC is null. The body terminal, and generally connected to that source (only three terminals). Drain and Source are symmetrical.

Applying a sufficiently positive voltage between the gate and source, the electrons are attracted towards the Si–SiO_2 interface under the gate forming a conductive channel between source and drain. In these conditions, if it is applied a $V_{ds} > 0$, a current may flow between the drain and source. The current between drain and source is controlled by the voltage between gate and source, which controls the formation of the channel. The electrical characteristics of the MOSFET depend from L (gate length) and W (gate width), as well as technological parameters such as oxide thickness and doping of body. Typical values of L and W are: L = 0.1–2 μm, W = 0.5–500 μm. The range of gate oxide thickness is 3–50 nm.

There are 4 types of MOS transistors: 2 to channel n and 2 to channel p. The MOSFET n-channel (nMOS) are formed on a p-type substrate:

- nMOS enrichment (enhancement) or normally off
- nMOS depletion (depletion) or normally on

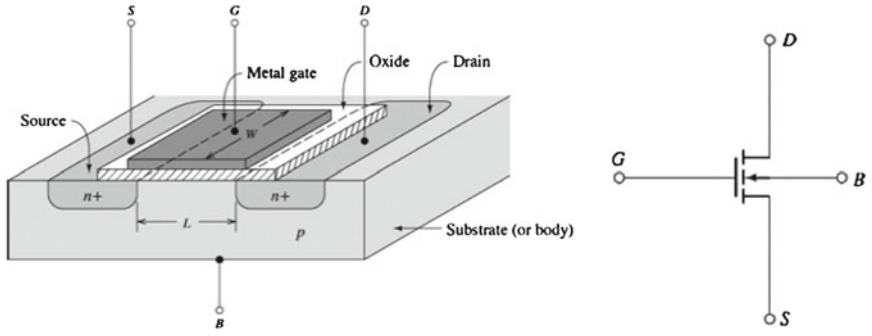


Fig. 1.14 Inside outline (general) and symbol of Mosfet

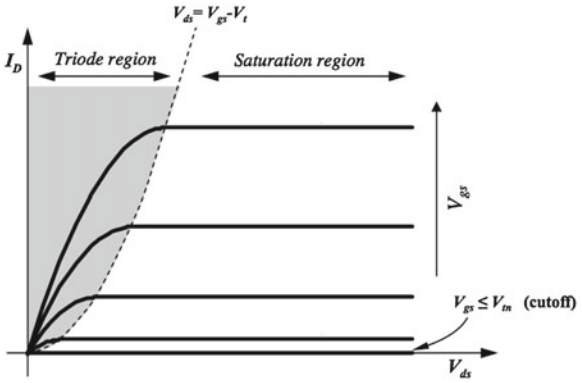
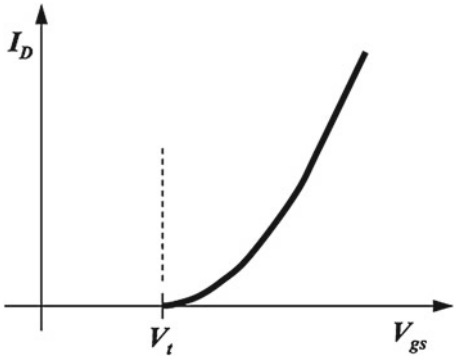


Fig. 1.15 Characteristic curve $I_d - V_{ds}$

Fig. 1.16 Saturation, $I_d - V_{gs}$



The MOSFET p-channel (pMOS) are formed on a substrate of n-type:

- pMOS enrichment (enhancement) or normally off
- pMOS depletion (depletion) or normally on

In most MOSFET applications, an input signal is the gate (G) voltage Vg and the output is the drain (D) current I_d. The ability of MOSFET to amplify the signal is given by the output/input ratio: the transconductance, $g_m = dI/dV_{gs}$.

In the Fig. 1.15 report the characteristic curve $I_d - V_{ds}$; there are three regions of work:

- Cutoff: In this case its necessary induce the channel, $V_{gs} \le V_t$ (V_t is the threshold voltage) for nMOS.
- Triode: The channel must be induce and also keep Vds small enough so the channel is continuous (not pinched off): $V_{ds} \le V_{gs} - V_t$.
- Saturation: In this mode need to induce the channel, $V_{ds} \ge V_{gs} - V_t$ then ensure that the channel is pinched off at the drain end. In Fig. 1.16 is visualized the plot of I_d versus V_{gs} for an enhancement type nMOS device in saturation.

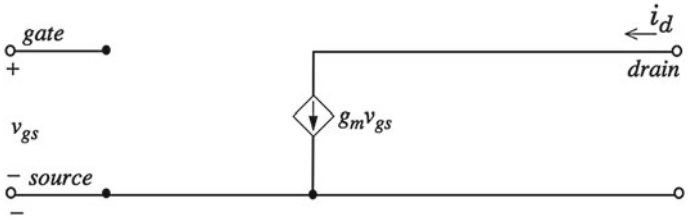


Fig. 1.17 Mosfet model (ideal)

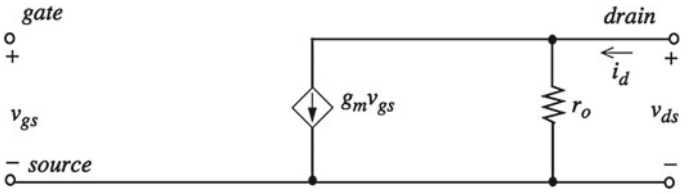


Fig. 1.18 Mosfet model (not ideal)

In the saturation mode, this device as an ideal current source (Fig. 1.17). In reality, there is to consider a finite output resistance (r_0); the outline of Fig. 1.18 can be described as in Fig. 1.17.

While the transconductance g_m gives the variations of the drain current due to variations of the voltage V_{gs}, there is another fundamental parameter of the Mosfet that takes the name of the output conductance, which is the variation of the drain current I_{ds} due to variations of the voltage V_{ds}.

The output conductance assumes a much greater importance, and the reason is the effect of the so-called 'channel length modulation' due to V_{ds}: it is the effect for which the effective length of the channel decreases with increasing V_{ds} and in the saturation zone, the current increases with the V_{ds}. From an analytical point of view the effect of the modulation of the channel length can be written as follows: $I_{ds} = k(V_{gs} - V_t) * 2 * (1 + V_{ds}\lambda)$. It is clear that provides a linear dependence of the I_{ds} versus V_{ds} in according to λ named as parameter of the channel length modulation; k is the transconductance factor proportional to the geometry of the Mosfet.

Example of application with Mosfet can be the common source Amplifier visualized in Figs. 1.19 and 1.20. The input signal is applied to the gate through the coupling capacitor C_1. The output is on the drain and connected to the load through C_2, C_1, C_2 and C_S are the coupling capacitors, and therefore can be considered short-circuits at the frequencies of the signal (center-band). The Source (S) is therefore to ground for the signals. R_1, R_2, R_D and R_S form a bias network to 4 resistors.

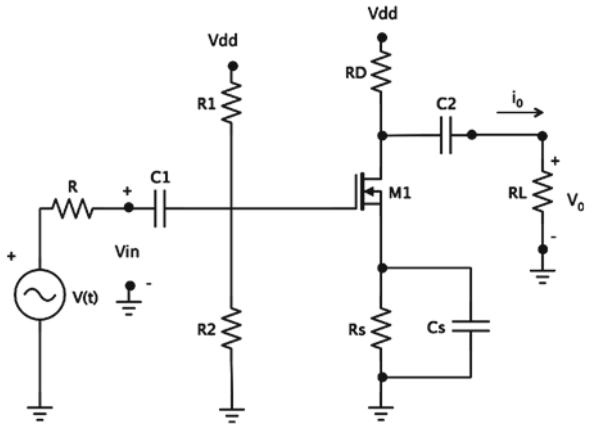


Fig. 1.19 Common source amplifier

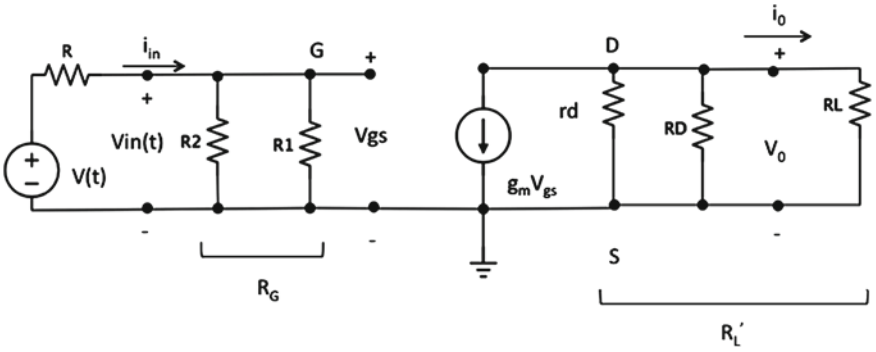


Fig. 1.20 Common source amplifier (model)

1.6 Differential Amplifiers

The main part of analog-integrated-circuit design is the differential-pair or differential amplifier configuration. The differential amplifiers were introduced in electronics to eliminate all or part of the problems of the amplifiers with direct coupling. The goal is to create amplifiers characterized by an acceptable signal/noise ratio (SNR), also in the presence of external/internal disturbances generated by thermal variations due to the aging of electronics components. The differential-pair of differential-amplifier configuration is widely used in IC circuit design. One example is the input stage of an op-amp and the emitter-coupled logic (ECL). This technology was invented in 1940 s for use in vacuum tubes; the basic differential-amplifier configuration was later implemented with discrete bipolar transistors. However, the configuration became most useful with the invention of the modern transistor/MOS technologies [3, 4].

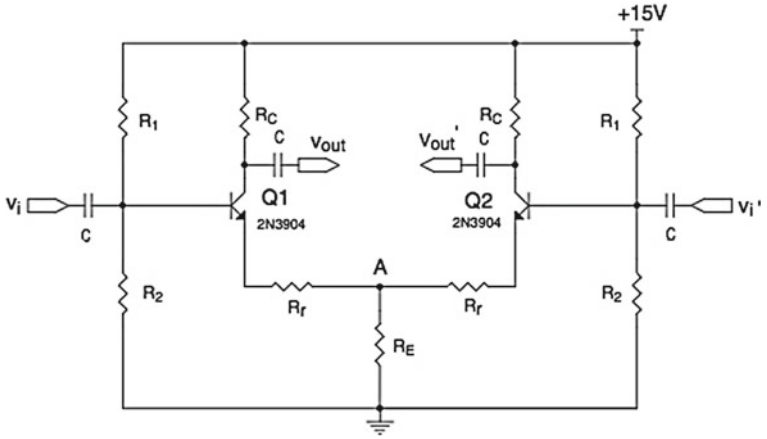


Fig. 1.21 Example of a differential amplifier

The main features of differential amplifiers are as follows: (1) High input resistance, so small voltage signals can be amplified without losses. (2) Temperature drift is minimal. (3) Two input terminals, i.e. inverting and non-inverting inputs. (4) It amplifies the difference between two input signals.

The differential amplifier works only on dual power supplies: it requires both $+V_{cc}$ and $-V_{cc}$ voltages simultaneously. However, if the same circuit is connected to a single power supply, its working becomes unstable and we do not get proper amplification from the circuit. The differential amplifier (Fig. 1.21) provides a number of advantages, making it one of the most useful circuit configurations, particularly as input stage for high gain and DC amplifiers. The figure of merit for differential amplifiers is called Common Mode Rejection Ratio (CMRR), defined as the ratio between difference mode gain (Ad) and common mode gain (Ac). Differential amplifiers is a special purpose amplifier designed to measure differential signals, otherwise known as a subtractor. The CMRR can be calculated with the following equation:

$$A_d = \frac{R_c}{2(R_r + r_{tr})}; A_c = \frac{R_c}{2R_E + R_r + r_{tr}}; CMRR = \frac{2R_E + R_r + r_{tr}}{2(R_r + r_{tr})} \quad (1.18)$$

where r_{tr} is the trans-resistance, usually indicated also as r_e. The 741 (a common op-amp chip) has a CMRR of 90 dB, which is reasonable in most cases. A value of 70 dB may be adequate for applications insensitive to the effects on amplifier output; some high-end devices may use op-amps with a CMRR of 120 dB or more. The common-mode rejection ratio (CMRR) relates to the ability of the op-amp to reject a common-mode input voltage. This is very important because common-mode signals are frequently encountered in op-amp applications.

Common application of differential amplifiers is in the control of motors or servos, as well as in signal amplification. In discrete electronics, a common arrangement for

implementing a differential amplifier is the long-tailed pair, typically found as a differential element in most op-amp integrated circuits. A long-tailed pair can be used as an analog multiplier with a differential voltage as one input and biasing current as another. There are many differential amplifier ICs on the market today. The AD8132 from Analog Devices (ADI) is a low-cost differential (or single-ended) amplifier with one resistor for setting the gain. The AD8132 is a major improvement on op-amps, especially for driving differential input ADCs or signals over long lines. The AD8132 can be used for differential signal processing (gain and filtering) throughout a signal chain, significantly simplifying the conversion between differential- and single-ended components. Linear Technology also provides many differential amplifiers for different applications. For example, its LTC6409 offers a very high speed and low distortion, and is stable in a differential gain of 1.

1.7 Feedback

Amplifiers can be considered not perfectly linear. The gain (or amplification) of the amplifier changes with power supply or temperature due to the variations of the working point of the transistors. These and other real limitations of the amplifiers can be minimized with the use of negative feedback. The good functionality of an amplifier and sometimes limited by the presence of extraneous signals, such as the hum of power supply, coupling with other amplifiers neighbors, etc. In some specific cases, the negative feedback can reduce these effects, while in other cases it does not induce any improvement. An important example are the amplifiers and hi-fi audio systems, which are powered with DC voltage obtained by rectifying AC power supply and not perfectly stable [3–5]. In Fig. 1.22 is shown a possible feedback outline. The output signal of the amplifier is always smaller than power supply voltages and its shape can be a clear demonstration that the output is not linear. The result of this non-linearity is manifested as a distortion of the output signal.

The use of negative feedback allows to realize amplifiers with better performance. The main effects of feedback loop can be the following:

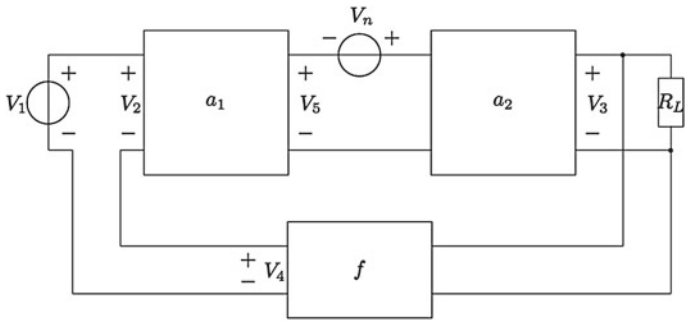


Fig. 1.22 Feedback outline

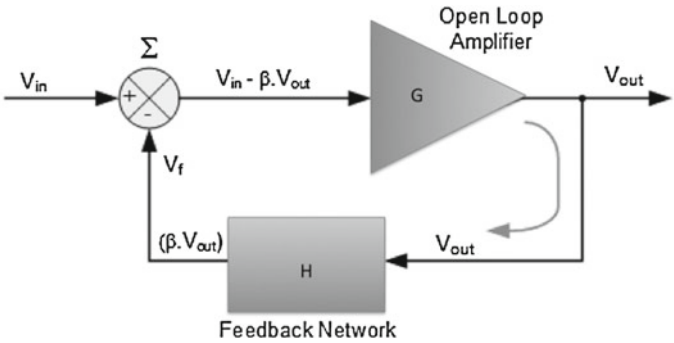


Fig. 1.23 Negative feedback outline (general)

- greater stability of gain
- less distortion
- greater bandwidth
- reduction of the noise

In general, the behavior of the individual block of negative feedback can be described by transfer function (Fig. 1.23).

The G block is called forward, the block H is the feedback: the output signal V_{out}(t) is sent back, through H, in input and is controlled by comparator. The difference signal represents the error of the system, it acts as a input signal to the block forward to have in output the corrected signal.

If the system is sufficiently fast, any changes of G block do not affect the output signal. More generally, if the input signal changes in time, the negative feedback system gives the necessary corrections to ensure that the output signal is a faithful repetition (with amplification) of the output.

Otherwise a positive feedback can be used to design oscillators. Positive feedback is responsible for the squealing of microphones when placed too close to the speaker through which their input signals are amplified.

When the loop gain is positive and above 1, there will typically be exponential growth, increasing oscillations or divergences from equilibrium.

1.7.1 Effects of Feedback

A disturbance or noise is an unwanted signal that is superimposed on the input/output of a circuit. Disruptions can occur in various parts of the system, their effect are greater in the input.

Some consideration about the effects of the noise in the loop systems:

- the open loop systems are not able to compensate for the disturbance but if noise is present at the output the effect is smaller because the useful signal which is superimposed has an amplitude greater.
- the closed loop systems are able to limit the effect of the disturbance when this is present in the output but not when it is present at the input.

The effect of disorder can be evaluate thought the ratio signal-to-noise (S/N or SNR). The time that elapses between the moment in which it has the effect and the moment in which this effect is taken into account to modify the system is called "delay in the feedback loop". When this delay is high, there may be problems of stability. The Nyquist criterion allows to investigate the stability of the closed loop system of the known transfer function in open loop and in particular its polar plot or Nyquist plot. The Nyquist plot of a closed system in feedback is a representation in the Gaussian plane of the value of frequency response function in open loop, G (jω), in terms of the real part and imaginary part, with the variation of the pulsation ω.

1.7.2 PID Controller

The PID controller is probably the most-used feedback control design. PID is an acronym for Proportional-Integral-Derivative, referring to the three terms operating on the error signal to produce a control signal. The desired closed loop dynamics is obtained by adjusting the three parameters: proportional, derivative and integral often iteratively by "tuning" and without specific knowledge of a plant model. Stability can often be ensured using only the proportional term [6].

1.8 Digital CMOS Circuits

It is virtually impossible to find electronic devices in our daily lives that do not contain digital circuits, and with most of them having CMOS logic devices at their heart. There are a large number of CMOS devices, divided in a number of families. Using only very few components, it is possible to build fairly elaborate pulse and signal generators [3–5].

Depending on the doping material used, there are mainly two types of metal-oxide-semiconductor field-effect transistors (MOSFETs): the n-channel, or nMOS, and p-channel, or pMOS. In N-type metal-oxide-semiconductor (NMOS) logic, n-type MOSFETs are used to implement logic gates and other digital circuits. These circuits are mostly used for switching due to their high speed nature, whereas PMOS circuits are slow to transition from high to low state, and their asymmetric input logic levels makes them susceptible to noise [3, 4]. However, metal-oxide-semiconductor

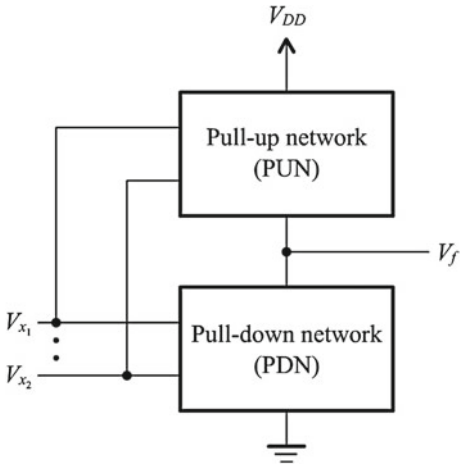


Fig. 1.24 The structure of CMOS circuit

(CMOS) technology offers some attractive practical advantages over NMOS technology: high noise immunity and low static-power consumption. CMOS uses a combination of p- and n-channel MOSFETs as building blocks, but here both low-to-high and high-to-low output transitions are fast since the pull-up transistors have low resistance when switched on, unlike the load resistors in NMOS logic. In addition, the output signal swings the full voltage between the low and high rails. This strong, nearly symmetric response also makes CMOS more resistant to noise. In NMOS circuits the logic functions are realized by arrangements of NMOS transistors, combined with a pull-up device that acts as a resistor. The concept of CMOS circuits is based on replacing the pull-up device with a pull-up network (PUN) that is built using PMOS transistors, such that the functions realized by the PDN and PUN networks are complements of each other. Then a logic circuit, such as a typical logic gate, is implemented as indicated in Fig. 1.24. It comprises a network of NMOS transistors (pull-down network) and a network of PMOS transistors (pull-up network), but each consisting of an equal number of transistors. Each input variable requires an NMOS transistor in the pull-down network and a PMOS transistor in the pull-up network.

1.8.1 CMOS Inverter

A CMOS inverter (Fig. 1.25) is composed of two MOSFETs, with their gates connected to the inverters input line, and their drains to the output line. The input resistance of the CMOS inverter is extremely high, as the gate of an MOS transistor is virtually a perfect insulator and draws no input direct-current. Since the input node of the inverter only connects to the transistor gates, the steady-state input current is nearly zero. A single inverter can theoretically drive an infinite number of gates (or have an infinite fan-out). A CMOS inverter dissipates a negligible amount

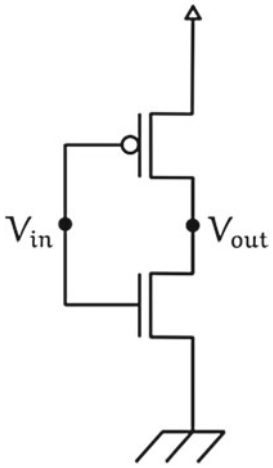


Fig. 1.25 CMOS inverter

of power during steady-state operation, which occurs only during switching. This makes CMOS technology useable in low power and high-density applications.

1.8.2 Example of Circuits

SRAM Cell
Static random access memory (SRAM) can retain its stored information as long as

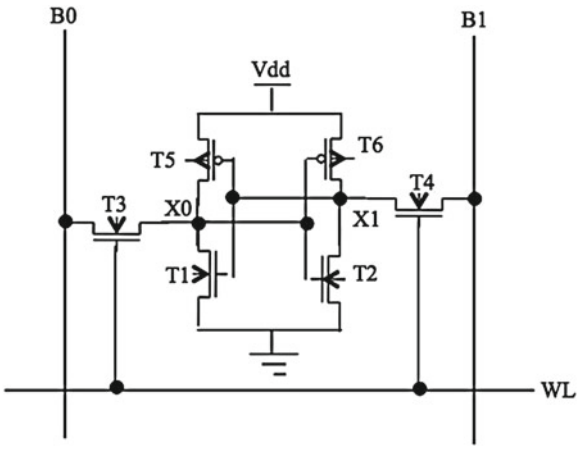


Fig. 1.26 SRAM cell

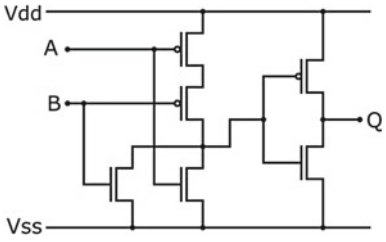


Fig. 1.27 CMOS OR

power is supplied. Its circuit schematic is shown in Fig. 1.26. It comprises two cross-coupled inverters (positive feedback), in particular four n-FETs and two p-FETs. The core of the cell is formed by two CMOS inverters, where the output potential of each inverter is fed as input to the other. This feedback loop stabilizes the inverters to their respective state. Access to the cell is enabled by the word line (WL); moreover, bit lines (indicated with B0-B1) are used to read and write from and to the cell.

CMOS OR

The OR gate (Fig. 1.27) is a digital logic gate that implements a logical disjunction: a HIGH output results if one or both inputs to the gate are HIGH. OR gates are basic logic gates and, as such, they are available in TTL (transistor-transistor logic) and CMOS ICs logic families. TTL is a class of digital circuit built from bipolar transistor (BJT) and resistors. TTL became the foundation of computers and other digital electronics.

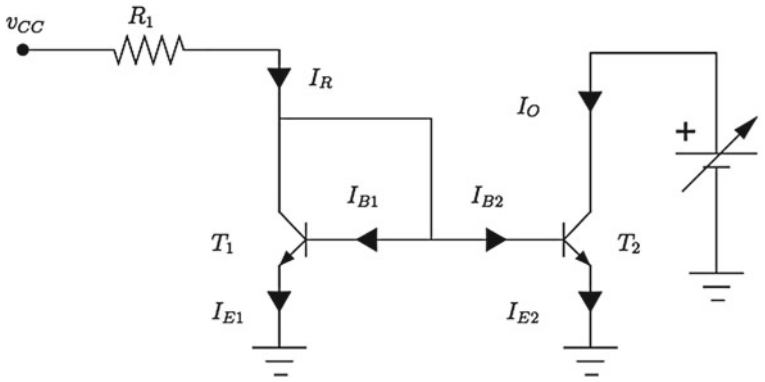


Fig. 1.28 Current mirror with BJT

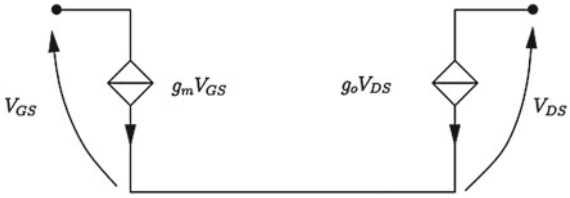


Fig. 1.29 Current mirror with mosfet (model)

1.9 Current Mirror

Current mirrors replicate the input current of a current sink or current source in an output current, which may be identical or a scaled version. Current mirrors are used to provide bias currents and active loads to circuits [3–5]. Apart from some special circumstances, a current mirror (Figs. 1.28 and 1.29) is one of the basic building-blocks of the operational amplifier; it is a circuit designed to keep the output current constant regardless of loading. This type of topology may therefore be used in order to create current generators. The main requirements a current mirror must meet are:

- Output current independence of the output voltage.
- Wide range of output voltages at which the mirror is working properly.
- Low input voltage.

The range of voltages within which the mirror works is called the 'compliance range', and the voltage marking the behavior in active/linear region is called the 'compliance voltage'. There are also a number of secondary performance issues with mirrors, such as temperature stability for example.

1.9.1 Ideal Current Mirror

A current mirror is usually and simply approximated by an ideal current source. However, an ideal current source is unrealistic for several reasons:

- It has an infinite AC impedance, whereas a practical mirror has a finite impedance;
- It provides the same current regardless of voltage, that is, there are no compliance range requirements;
- It has no frequency limitations, whilst a real mirror has limitations due to the parasitic capacitances of the transistors;
- The ideal source has no sensitivity to real-world effects like noise, power-supply voltage variations and component tolerances.

The main part of the current mirror is a bipolar transistors (BJT) or a MOSFET. Transistors in a current mirror circuit must be maintained at the same temperature for precise operation. There are different types of current mirrors:

- Simple current mirror (BJT and MOSFET);
- Base current corrected simple current mirror;
- Widlar current source;
- Wilson current mirror (BJT and MOSFET);
- Cascode current mirror (BJT and MOSFET).

All of the circuits have compliance voltage that is the minimum output voltage required to maintain correct circuit operation: the BJT should be in the active/linear region and the MOSFET should be in the active/saturation region.

1.9.2 Current Mirror BJT/MOS

Current mirror circuits are usually designed with a BJT, such as an NPN transistor, where a positively doped (P-doped) semiconductor base is sandwiched between two negatively doped (N-doped) layers of silicon. These transistors are specifically designed to amplify or switch current flow. In some current mirror design specifications, the NPN transistor works as an inverting current amplifier, which reverses the current direction, or it can regulate a varying pulse current through amplification to create output mirror properties. One of the reasons that BJTs are used for current mirror design is due to the base-emitter (or PN part) of the transistor functioning reliably like a diode. Diodes regulate both the amount of current that passes as well as the forward voltage drop for that current. The basic current mirror can also be implemented using MOSFET transistors (Fig. 1.30). In Fig. 1.30, M_1 is operating in the saturation or active mode, and so is M_2. In this setup, the output current I_{OUT} is directly related to I_{REF}. The drain current of a MOSFET I_D is a function of both the gate-source voltage and the drain-to-gate voltage of the MOSFET given by a relationship derived from the functionality of the MOSFET. In the

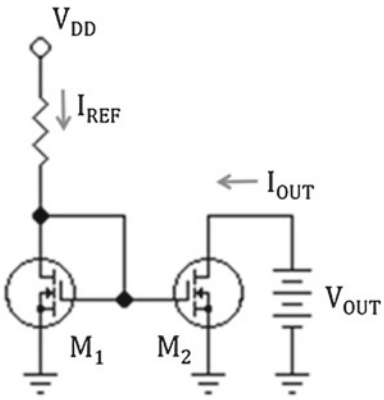


Fig. 1.30 Current mirror with mosfet

case of transistor M_1 of the mirror, $I_D = I_{REF}$. Reference current (I_{REF}) is a known current and can be provided by a resistor or by a 'threshold-referenced' or 'self-biased' current source to ensure that it is constant and independent of voltage supply variations [7].

References

1. Park, J., & Mackqy, S. (2003). *Pratical data acquisition for instrumentation and system control*. USA: Elsevier.
2. Lacanette, K. (2003). National temperature sensors handbook. *National semiconductor*. USA: National Semiconductor Corporation.
3. Razavi, B. (2008). *Fundamentals of microelectronics*. UK: Wiley.
4. Kang, S. M., & Leblebici, Y. (1998). *CMOS digital integrated circuits*. New York: McGrawHill.
5. Razavi, B. (2002). *Design of analog CMOS integrated circuits indian*. New York: McGrawHill.
6. Wikipedia http://en.wikipedia.org/wiki/PID_controller
7. Wikipedia http://en.wikipedia.org/wiki/Current_mirror

Chapter 2
Features of Embedded System

Abstract This chapter will introduce the basic elements of embedded systems (or dedicated systems). The integrated control systems represent one of the areas of modern electronics which most important and developed with a variety of application in many fields from biomedical to the automotive.

2.1 The Components of Embedded System

An embedded system (Fig. 2.1) is, basically, a computer controlled device designed to perform specific tasks. In most cases, these tasks help revolve the real-time control of machines or processes. Embedded systems are cheaper than general purpose system, such as PCs [1, 2].

2.1.1 Processor

The main part of an embedded system is the processor, which could also be a generic microprocessor or a microcontroller and programmed to perform the specific tasks for which the integrated system has been designed.

2.1.2 Memory

Electronic memory is an important part of embedded systems and three essential types of memory can be described: RAM, or random access memory, ROM, or read only memory, and Cache. The RAM is one of the hardware components where data are temporarily stored during execution of the system. The ROM contains input-output routines that are needed for the system at boot time. The cache, instead, is used by the processor as a temporary storage during the processing and transferring of data.

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_2

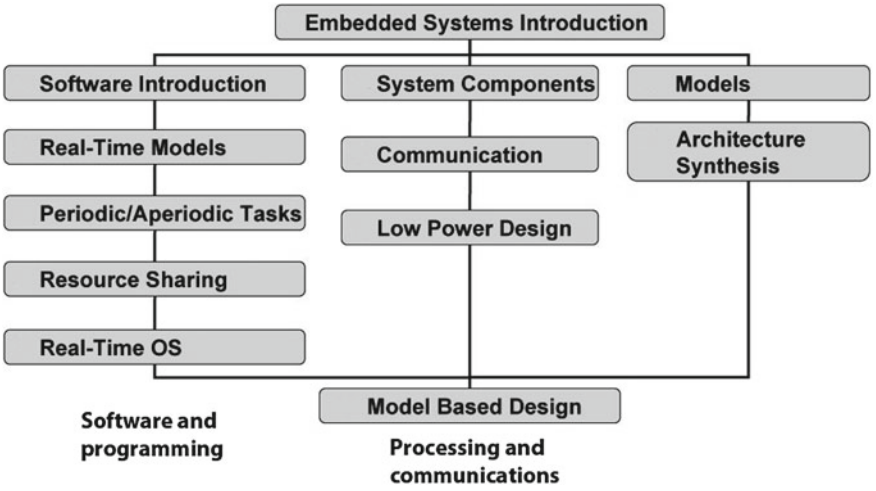


Fig. 2.1 Embedded system design

2.1.3 System Clock

The system clock is used for all processes is running on an embedded system and requires precise timing information. This clock is generally composed of an oscillator and some associated digital circuitry.

2.1.4 Peripherals

The peripheral devices are provided on the embedded system boards for an easy integration. Typical devices include serial port, parallel port, network port, keyboard and mouse ports, a memory unit port and monitor port. Some specialized embedded systems also have other ports such as CAN-bus.

2.2 Characteristics and Example of Embedded System

Most embedded systems are designed to perform an continued action at a low cost. Most of these systems also have constraints on the performance in terms of hardware and software, such as require operating in real time when a system needs high sped while executing some functions, but may tolerate lower speed for other activities. It is difficult to characterize the speed or the cost of an generic embedded system, especially for systems that have to process a large quantity of data. Fortunately,

most of the embedded systems have the essential characteristics that can be designed with a combination of hardware and high-performance software. To get an idea, just think of a decoder for a satellite television. Although a system should process tens of megabits of data per second, most of work performed by dedicated hardware, separates rule and decoding of the data flow in multi-channels digital video output. Embedded CPU calculated the locations of the data in the system, manages interrupts and clock systems [3, 4].

Usually, the hardware of an embedded system must comply with the performance requirements much less stringent in according to the hardware of the primary system itself. This allows that the architecture of an embedded system, for example, must be intentionally simplified and compared to that of a general-purpose computer with the same tasks, using a CPU more economic that basically behaves well for these secondary functions.

In the case of portable systems, costs reduction becomes a priority. This kind of system, in fact, often are made by a highly integrated CPU, a chip dedicated to all other functions and a single board of memory. In this case each component is selected and designed to reduce as much as possible the costs. The useful software to manage many embedded systems is called firmware. The firmware is a type of software that, for example, can be found in ROM or Flash memory chips. The software and firmware are designed and tested with much more attention than traditional software for personal computers.

Many embedded systems avoid incorporating components with moving parts (such as hard disk) that are less reliable than solid-state components such as flash memory.

Moreover, embedded systems may not be physically accessible (e.g. space systems); therefore, the system must be capable of a self-reset in case of data loss or corruption. This feature is very often obtained with the addition of a component called Watchdog that resets the computer in regular time intervals by an internal timer.

In the design of a modern and reliable embedded system is possible to note two fundamental characteristics: reprogrammability and the dimension. In fact, it would be helpful to think of a dedicated system can be readapted if, for example, system upgrade is required. Embedded systems are a classical discrete elements of ASIC design that allow the advanced optimization because the hardware occupies the necessary space strictly, making the control system easily integrated. An application-specific integrated circuit (ASIC) is an integrated circuit (IC) that has been customized for purpose use. Today the control of vehicles is one of the main applications of embedded systems. In a single high-end car you can find hundreds of embedded systems called ECU (Electronic Control Unit), physically distributed in the vehicle and connected to the different internal networks (networks intra-vehicle) specially designed, in most cases with stringent requirements of ‘quality of service’. A computer is the first and foremost versatile: it can be programmed to suit various areas of application. Conversely, the embedded system is a device dedicated to the performance of a single task, or a very narrow class of tasks. Thanks to the specificity of the run application, the embedded system can be designed to optimize particular

of cost and performance. The general purpose of the computers is designed with standards and reference architectures; and vice versa is difficult to define standards for embedded systems because each specific application leads to different design choices. Typical functions of embedded systems can be the following:

- Processing: ability to process the analog/digital signals.
- Communication: ability to transfer signals ("Information") from/to the outside world.
- Storage: the ability to preserve the temporary information within the embedded system.
- Each specific application made by an embedded system has different requirements for processing, power suppy, storage and communication.
- A same functionality (e.g., the ability to acquire still images via a CCD sensor) can be optimized radically in different way when applied, for example, a digital camera or a cell phone or a digital camcorder.

Morever, commercial features of embedded systems can be described in the following points:

- Final Cost: The cost of the final product is a very important parameter for the design choices.
- Time to market: in the design of an embedded system must always keep in mind the timing you want the product listed on the market; taking too long to design a device means that it's difficult to overcome the fast changes in the market.
- Life time: Another important factor is the expected lifetime for the product; which can varied from a few days to several years or decades.
- Volume: the quantity of stock planned for the system is one important factor in the design phase.

Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. Hardware and software characteristics of embedded systems can be described in the following points:

- Communication interfaces: typically the sale price of an embedded system is low, the choice of communication interfaces is critical because it greatly affects the final price of the product.
- User Interface: In many embedded systems the user interface consists of a few buttons and/or LEDs; in others, it uses the user interface of a host system.
- Power management: is a crucial factor to be considered for all embedded systems are powered by batteries.
- Dimensions and weight: in many cases, the physical characteristics are another critical factor; usually the embedded system must be small, very light or with a particular form (for example, very thin).
- Quality of service: many applications of embedded systems have stringent requirements in terms of QoS (Quality of Service); as a particular case, many applications require the provision of services in real time with stringent timing constraints.

- Code size: the storage capacity of embedded systems is limited, so the size of the internal program (e.g. firmware) is a important factor.
- Numeracy/Communication skills/Storage: commensurate to the specific application performed by the embedded system.
- Updating the program: it is useful to include the ability to update the programs in embedded systems so as to correct errors discovered after the production and introduce new features.

In addition to the parameters involved in the market, the hardware, software features and embedded systems are used to be dependable. Actually, in the design phase is necessary to consider the following aspects:

- Reliability: realistic assessment of the probability that the system fails.
- Maintainability: the system can be repaired or replaced within a certain time interval.
- Availability: probability that the system is working; essentially depends on the reliability and maintainability.
- Safety: properties related to the possibility that in the event of system failure are caused damages to people or things.
- Security: resilience of the system against unauthorized use.

To design the embedded systems, it should take into account aspects such as the speed of development, the economy of scale, maintainability and so on. Consequently, it is not possible to develop the hardware also without consider the software design. If the embedded system is safety-critical, the choice of the software organization plays a crucial role in the ability to certify the system for the use to which it is intended.

The real-time system is a system designed to operate within the well-defined time parameters. Practically, a real-time system operates correctly only if every input configuration is produced by the right output respecting well-defined time constraints.

2.3 Hardware and Software Design

The device required to achieve by designing an embedded system is, certainly, a system that will optimize various metrics of design; the most common are as the following [1–4]:

- Unit cost and cost NRE (non-recurring costs).
- Size and weight.
- Performance and power consumption.
- Flexibility and maintainability.
- Time-to-market.
- Correctness.
- Security of the system.

Usually, these metrics are in contrast between each other and become necessary to choose a compromise, esteeming and considering in an accurate way. Crucial importance is the metrics of cost/performance/power consumption and those of time-to-market/NRE. The relationship between time-to-market and NRE is very crucial to avoid entering in the market too late; If you have knowledge that the units produced will be few, it will be preferable to have lower NRE costs because it will not be possible to cover the next phase of the sale. It is essential to design an embedded system based on the constraints imposed by the metric, for this we use a methodology that uses estimators and cost modeling.

The constraints required for certain applications are more and more stringent, response times, size, weight and low power consumption. The flow of prototyping is often very complex and becomes important the reuse of pre-designed and tested hardware/software components is to reduce the time-to-market and NRE costs.

There are many independent techniques for estimating the costs: analogy, top-down, BottomUp, parametric; is preferable to use more than one and combine the results. Software parameter models (COCOMO) and Hardware (FPGA) allow to estimated the metrics automatically. The applicable cost to the modeling and design the process is called RASSP (Rapid Prototyping of Application Specific Signal Processors) and can use different methodologies.

The platform on which an embedded system can be developed varies drastically and depends on its complexity, utilities (electrical), cost, and scope of use: going from the PLC and microcontrollers to more complex architecture based on sophisticated integrated circuits (System-on-a-chip, SoC). The SoC enclose, in a single ASIC integrated circuit, microcontroller/CPU and/or DSP, memory, and clock oscillator, voltage regulator, any interfaces A/D, D/A, and external port (USB, ethernet, etc.). Some development platforms (reference board and reference design) are widely used based on ARM, MIPS, Coldfire/68k, H8, SH, V850, FR-V, M32R, and so on, for example IBM-compatible architectures with X86 or PowerPC CPU. Other architectures are based on PICmicro, Intel 8051 and Atmel AVR microcontrollers. Another common design method involves the use of FPGAs (Field-Programmable Gate Array), with the programming of all the internal logic, including the CPU. Typically for interfacing you will use the FPGA with other integrated circuits. This situation is in contrast with desktop computer market, which currently consists of only a few competing architectures, mainly the Intel/AMD X86 and PowerPC Apple/Motorola/IBM.

It's also useful to disclose the standard PC/104, dealing only with the form factor (the size of the motherboard) and the communication bus. It is typically employed in the industrial desktop systems (X86 CPU) with adaptations for specific uses. The user interfaces for embedded systems are also various between systems and therefore deserve some additional comment.

The designers of interfaces such as Apple Computer and HP tend to minimize the number of different user interactions. For example, their systems using only two buttons (the absolute minimum) to control a menu system. A touch screen or buttons to the edges of the screen can be also used to minimize the interaction with the user. As with other software, embedded system designers use compilers, assembler and debugger for developing the software supplied in the system.

An in-circuit emulator (ICE) is a hardware device that replaces or interfaces with the microprocessor, and provides functionality to load and debug test within the system. To speed up, make diagnostics and debugging, especially on large scale systems manufactured, is integrated at the level of SoC, microcontroller, or CPU, a JTAG interface (IEEE 1149.1): a standard simple interface and inexpensive that suspends the normal mode of the process and interrogate the phases by connection to a personal computer.
Some utilities add a redundancy check (checksum) or a Cyclic Redundancy Check (CRC) to the program, in order to allow the embedded system to check the validity of the program.
For systems using Digital Signal Processor (DSP), designers can use a tool such as MathCad or Mathematica to simulate the mathematics.
Compilers and specific linker can be used to improve the optimization of hardware. An embedded system may have its own specific language or development program or offer improvements to an existing language.
The programs for the generation of the software may have different origins: companies specialized in embedded system market, GNU project, development tool for personal computer if the embedded processor is very similar to a common PC processor.
The presence or absence of a complete operating system on an embedded system depends on its architectural complexity and the range of use. In the simplest case the embedded devices might be without an operating system itself. On simple microcontrollers typically operate cyclically few bytes of a single program, sometimes referred as a program of "monitor", dedicated mainly to monitor the status of the ports I/O. Complex environments can be applied to the same operating systems commonly used for the general purposes (Linux, Windows CE etc.), possibly custom (to operate in an environment with minimal resources), or more specialized to handle events of real-time operating systems (such as VxWorks and QNX), or highly specialized and not available on the market.

References

1. Heath, S. (1997). *Embedded systems design*. Oxford: Newnes.
2. Vahid, F., & Givargis, T. (2002). *Embedded system design: A unified hardware/software introduction*. Hoboken: Wiley.
3. Wilmshurst, T. (2008). *Designing embedded systems with PIC microcontrollers*. Oxford: Newnes.
4. (2008). *The art of designing embedded systems*. Oxford: Newnes.

Chapter 3
Microcontroller Design

Abstract This chapter will introduce the characteristics of microcontrollers; in particular above all, we analyze the hardware and software tools for implementing in embedded systems.

3.1 Introduction

The first microcontroller 4004 introduced by Intel in 1971, since then the growing of demand began for the fabrication of the implementation of the electronic systems of automatic control devices. The contemporary TMS1802 by Texas Instruments, designed for computers, was also completed in 1971 and used in cash registers and in different measuring instruments.

The first device to widespread in industrial level was the Intel 8048, which was integrated into PC keyboards. His success, the Intel 8051 as well as the series of Motorola 68HCxx, is still the standard for designers. Currently the production of microcontroller is in the billions of dollars annually and integrated in many systems, such as the following:

- Appliances such as microwave ovens;
- Telecommunication systems;
- Automotive Industry
- Aerospace Industry

Before the description about the basic architecture, it is necessary to clarify the differences between microcontroller (Fig. 3.1) and microprocessor. Although, both devices realize function of management and control of complicated system, the microprocessors are not able to operate in stand-alone. Furthermore, microprocessors require memory and an external interface I/O in order to realize a control system. For those, is often preferred to use a microcontroller, which basic scheme is shown in (Fig. 3.1); in addition to a central computing unit, a microcontroller contains all those management peripherals able to make a control system self-sufficient [1, 2].

A microcontroller (MCU or MC-MicroController Unit) is a programmable logic device, usually made with a static CMOS process, that combines electronics components integrated into a single chip: CPU, memory and peripheral devices, I/O.

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_3

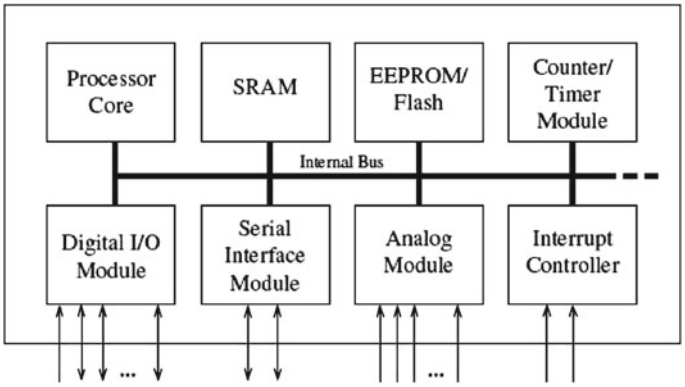


Fig. 3.1 General outline of a microcontroller

The connections between the CPU and any other functional block inside the microcontroller are made through the bus, in which data and addresses of memory locations can circulate in according to the three different types of buses:

- Control bus: the set of all control signals exchanged by the CPU with memory devices, data processing, inputs/outputs that coordinate and synchronize the activities of the system.
- Address bus: the addresses that corresponding to a memory location or a device input/output.
- Data bus: it moves data stored in a memory location (previously addressed thanks to the address bus) to the CPU or vice versa.

The three buses can be physically separated or 'multiplexed', i.e. some bus lines take on different functions at different times, so it must be a hardware that provides appropriately to separate and use the information and functionally different present on the same lines. Regarding the number of lines that established the various buses, it depends on the characteristics of the system. In particular, the embedded microcontroller realized for flash memory has both the data bus and the address bus of 16 bits.

There are two main microcontrollers families: CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer). CISC are for general purpose processors, use the Von Neuman architecture (e.g. in the ST10 microcontroller), characterized by a single bus between the CPU and memory, and a common memory for data and instructions; it involves more than one machine cycle for the execution of each instruction, this constitutes a first disadvantage. Another disadvantage of CISC microcontrollers is that you can not use clock signals in high frequency.

RISC microcontrollers, instead, are Harvard architecture (e.g. in the ST40 microcontroller), which provides the data memory separated from the program memory, two separate buses between the CPU and data-memory, and between the CPU and program memory.

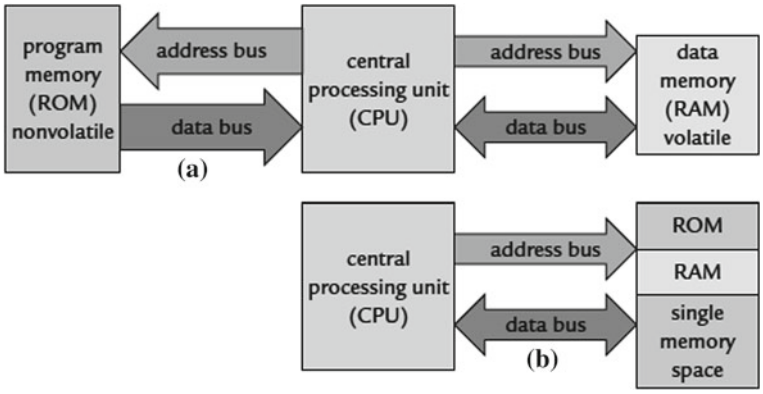


Fig. 3.2 Harward (**a**) and Von Neuman (**b**) architecture

Then, the CPU can perform two parallel entrances, using techniques of pipeline to execute an instruction in every machine cycle. The few numbers of RISC instructions and addressing mode make this architecture more simple and convenient than RISC. Some features of RISC architecture can be described in the following:

- More compact,
- Higher frequency,
- Execution of instructions in a single clock,
- Length of instructions is kept constantly in order to ensure predictability in the time of execution of the instructions.

The RISC design philosophy is to provide a unit, or the intelligent part of the system, with a set of registers that allow the movement of data and instructions loaded from memory across internal registers. For the overall simplicity and its advantages, the industry tends to use microcontrollers of RISC architecture (Fig. 3.2).

3.2 CPU

The basic schematic of a CPU is shown in (Fig. 3.3). It is composed essentially of two main units: the data path which executes instructions, the control unit which manages the data and instructions [3, 4].

The CPU tasks are to analyze/execute the instructions and read the data from memory. The result of an instruction executing depends on the data on which it operates and the internal state of the CPU itself. It is possible to distinguish two types of architectures for CPUs: the classical Von Neumann architecture in which data and instructions reside in the same memory; and the Harvard architecture, which thanks to both the division of the data bus and address bus that allow the CPU to work faster. Main elements of the CPU are as the following:

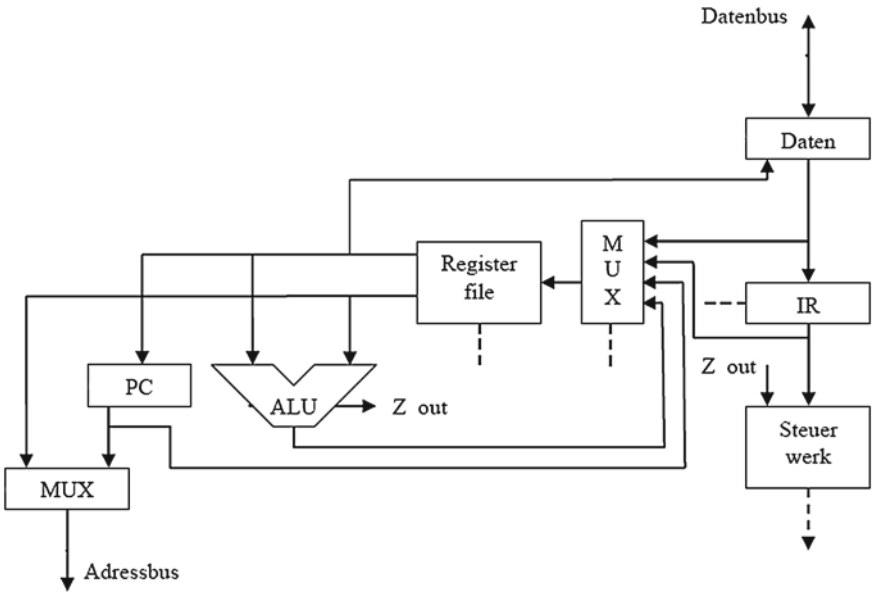


Fig. 3.3 General outline of a CPU

- ALU (Arithmetic Logic Unit): it is responsible for performing arithmetic operations;
- Control Unit: it manages instructions and data, executes the statement and the result is stored in special registers (with a number of bits that depends on CPU architecture);
- Generic registers: They have a high access speed and depend on the complexity of the processor.

In each CPU we can define the following special registers:

- registroPC (ProgramCounter): contain memory address of next instruction to be executed;
- register CPSR (Current Program Status Register): It doesn't contain data but a set of flags which are marked with special states of the CPU;
- register SP (Stack Pointer): It is capable of handling the stack memory, used to call subroutines, temporary storage of data and use of interrupts. The SP register is automatically incremented or decremented. The address contained in SP is equivalent to the first free location in the stack.

Each CPU owns ISA (Instruction Set Architecture) instructions. The approach of the first CPU, dictated by purely economic and technology reasons, is leaning toward a set of instructions to make even complex operations. This approach, called CISC (Complex Instruction Set Computer), was soon supplanted by more modern RISC (Reduced Instruction Set Computer). Then, the role of the compiler, in this type of

CPU, plays a key role, to change the complex instructions into simpler instructions. The reduction of complexity in this type of processors has led designers to develop the concept of a pipeline. The pipeline can be interpreted as an assembly line, where each stage is responsible to perform a particular function. A single instruction needs to be done, at least three steps: fetch, decode and execute. The problem with this approach is the conditional jump instructions: the CPU could not know in advance whether or not it will have to perform the jump before you run the previous ones. So it must decide whether to set the pipeline taking account of the jump or not: in case of wrong prediction pipeline must be emptied completely and the instructions in the execution of decoding re-read by beginning, losing a number of clock cycles directly proportional to the number of stages of pipeline.

To avoid this, modern processors have drive-backhoe (Branch Prediction Unit) whose purpose is groped to predict whether, given a conditional jump instruction and those previously made the jump will be executed or not.

3.3 Memory

The memory in a microcontroller is an essential element; it allows storing the program to be executed and the useful information to the CPU on the device configuration. It is possible to divide the memory into two main categories: volatile and non-volatile. Usually in a microcontroller are present both types of memories and is usual to use Flash memory to write two fundamental information: the boot loader and the main program [4, 5]. The boot loader is the equivalent of the BIOS for the personal computer used from CPU to initialize the system. It is customary to write the boot loader in special reserved addresses of flash memory (usually larger than a dozen Kbytes).

The act of writing information is so important with advantages but also hazards. If you were accidentally to erase the boot loader, the system would not work properly.

On the other hand, the fact of being able to rewrite, involves a greater freedom for the designer, such as the possibility to perform the programming of the device by USB port rather than serial port. Furthermore, in the flash memory is normally also written the program to run. The portion of the dedicated memory can have variable dimensions and will depend on the particular application of the microcontroller.

RAM memories are, as mentioned, in the CPU itself in the form of dedicated registers. A microcontroller usually also contains a set of RAM but not for storing temporary information from external devices.

3.4 Devices

Within a microcontroller are indispensable devices for the management of the input signals. It's normal so that you have one or more counters and timer interrupt controller. The processing that needs to be able to make a microcontroller with the

system must not only be of a mathematical nature, but may range in a myriad of applications [1, 2, 6–8]. It is usual therefore to introduce into a microcontroller some devices: timer, A/D and D/A converters for a more flexible management of analog signals. Furthermore, a microcontroller must be able to react while events interrupt. The interruption allows intercepting a timer event, to pause the program and run a portion of a program specialized for handling the event occurred and resume the execution of the main program. There are two bits that can regulate the activity of interrupts: interrupt enable (IE) and the interrupt flag (IF). The interrupt enable is settled by the program to indicate that the ISR (Interrupt Service Routine) can be activated in response to an event. Instead, the IF flag is activated by event and resettled when the interrupt routine starts. Today, all controllers have a watchdog timer to monitor the software. The basic idea is the following: once it is activated, the timer starts inverse counting by a fixed number settled by the programmer. When the count reaches zero, an interrupt pulse resets the system avoiding that any program block may occur permanently. With additional hardware unit and features, customary microcontroller allows performing remote debugging of the chip from the PC.

3.4.1 I/O Devices

The peripheral I/O devices allow the communication with the outside world. The main interface system can be the SCI (Serial Communication Interface) and UART (Universal Asynchronous Receiver Transmitter). In (Fig. 3.4) the basic structure dell'UART is show.

It is not a protocol but a form that can be used for the asynchronous communication: uses two lines, a transmission and a reception, to obtain a half or full duplex communication. In a normal microcontroller, the UART module has four types of configurable parameters: number of data bits, parity, stop bit and baud rate. The most famous physical interface without a doubt is the RS232 interface [1, 2, 6–8].

Peripherals I/O present in almost all microcontrollers are the IC2BUS, 2-wire serial interface developed for applications with 8 bits; the PWM (Pulse Width Modulation), a constant frequency pulse generator with variable duty-cycle is often used as a digital/analog converter; the CAN (Control Area Network), a protocol developed in the early 80s used initially within the automotive industry. For CAN, a single pair can be used for the exchange of information between the different devices and control units on board.

JTAG (Joint Test Action Group) is another interface used in each microcontroller for debugging purposes. IT uses four pipes to enter in serially way within a chain of the microcontroller internal registers which able to test the main features of the device. In the most modern microcontrollers are integrated USB controllers: having a protocol more complex certainly, but also have more contemporary than the classic serial interface, USB communication has not yet become a standard in the field of microcontrollers and is not used to its full potential, neither.

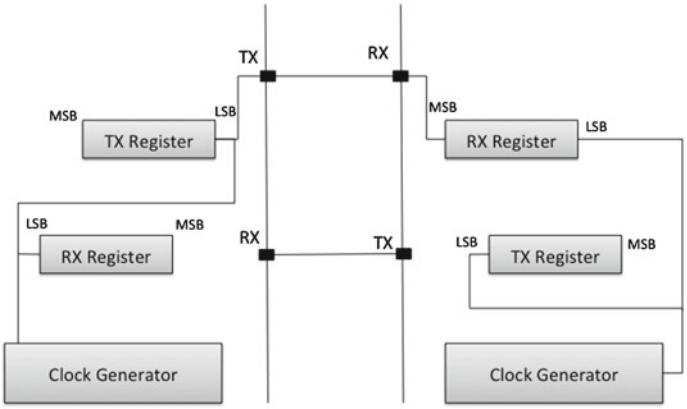


Fig. 3.4 General outline of UART

3.5 Power Saving

The microcontroller can be programmed to operate in a special mode to lower the power consumption:

- Idle mode (also called sleep-mode): all activities are stationary, except for the circuitry that relates the oscillator, watchdog timer and Idle. The consumption in this mode is about 30 % of that in normal mode of operation. At regular intervals the Idle timer generates the awakening of the microcontroller (wake-up) to test that there are no problems, then returns to Idle. Following a reset or interrupt can wake the device to operate in normal mode and perform operations, after that return back to Idle.
- Halt mode: the mode is at lowest power consumption, where all the circuitry is inactive, including internal oscillator and timer. Only a reset or an interrupt can wake up the MCU.

In both cases, however, both the contents of RAM and the state of the outputs remains unchanged, for which the device does not change its internal state overall [1, 2, 6].

3.6 Instructions

The set of instructions executed by the Embedded Microcontroller, but in general, as well as any other microcontroller, are the instruction set and can be classified into major groups, in particular:

- Arithmetic and logic instructions;
- Loading instructions (LOAD) and transfer (MOVE) data;

- Jump instructions (JUMP, CALL, RET);
- Instructions WAIT;
- Instructions Interrupt;
- Instructions SET CLEAR PULSE.

Generally speaking, the logic provides operations on data or signals of AND, OR, NOT, SHIFT, MIRROR; while the arithmetic instructions include operations such as ADD (sum), SUB (subtraction), MULT (multiplication), DIV (division). The LOAD instructions can be distinguished as the following: loading a constant into a register, loading the contents of a register in the RAM or vice versa, loading a byte into a register and loading a word (16 bits) in a register. Branch instructions require that the Program Counter instead of pointing to the next instruction, points to a different memory address where begins the subroutine (JUMP, CALL) or where it return back to the main program (RTS Return from subroutine) [7, 8].

The JUMP and CALL instructions can be absolute conditioned or unconditioned, while the BRANCH instruction at the jump is executed only if, at the beginning, a particular condition is verified. The test may be of interest for an internal conditions, such as the logic state of the flags status (Carry, Overflow, Zero, Sign, Parity). In fact, these jump instructions can be encoded as instructions for loading an appropriate word on the 16-bits Program Counter; in the case of RTS instruction, will be loaded on the PC word contained in the stack pointer, which points to the first memory location of the Stack. This interpretation of the jump instruction, such as instructions for loading word, is exploited in the new instruction set for Embedded microcontrollers.

Regarding the interrupt system, there are instructions for IE (Interrupt Enable) and DI (Disability Interrupt) to enable and disable (or mask) interrupts; the instruction RETI (Return from Interrupt) is an instruction of RESET for switching and managing of priorities among the interrupts.

For each microcontroller, instructions can be encoded on one or more ‘word’ composed of two parts:

- The opcode: the type of operation;
- Operand Field: shows the registers, the addresses of the memory locations or data involved in the operations.

3.6.1 Enforcement of Instructions

The operation of the microprocessor is based on processing by the CPU of two phases:

- FETCH, characterized by the Operation Code Fetch: decode from the Operation Code and Fetch of the Operand Address;
- EXECUTE: executing instructions and storing of the results.

During the FETCH, the CPU loads the bus address, which is provided by the Program Counter (PC); on the control bus there are information to read the memory

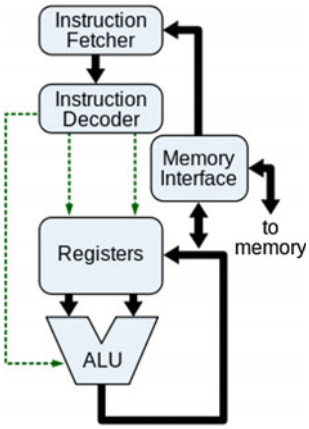


Fig. 3.5 Enforcement of instructions

location, while data is loaded on the data bus in according to the Instruction Register (IR) pointed by the program Counter. During the EXECUTE, instructions are loaded and performed in the IR, transferring the necessary data from memory to data-path-register, and the logical and/or arithmetic decoding derived from the operation code. The result, in according to the type of operation performed, is the rewritten in a register or a memory location or on a device I/O.

Normally, an instruction to be executed requires at least 2 machine cycles, since the phase of Fetch requires one cycle and the phase of Execute another one, with at least two memory accesses, one for reading and one for writing (Fig. 3.5) [1, 2, 6–8].

3.7 ARM Architecture

The ARM architecture (Advanced RISC Machine) is a family of 32-bits RISC microprocessors used in a multitude of embedded systems. Thanks to its low-power performance, ARM architecture dominates the field of mobile devices where the battery energy saving is critical [1, 2, 6–8].

Currently the ARM family covers 75 % of the world market, used in cell phones, media players, portable games and devices for computer. Important branches of the family are the ARM XScale processors and OMAP processors manufactured by Texas Instruments. The devices have ARM RISC architecture ‘load and store’ for 16 and 32 bit words, an orthogonal instruction set, 37 registers of 32-bits integers (6 status registers and 31 general purpose) and 7 operation modes (USR, FIQ, IRQ, SVC, ABT, SYS, UND). To make the project simpler than processors as the Intel 80286 and Motorola 68020, the processor included some unique features such as the following:

- Conditional execution of many instructions to reduce skips and stalls compensate for the pipeline;
- Arithmetic operations are the only ones that can alter logs of conditional executions;
- Shifter 32 bits that can be used in conjunction with other instructions without charge time;
- Addressing method;
- 2 levels of Interrupt very fast and simple with a switch subsystem of directors connected each other.

One of the most interesting part of the ARM processors are 4 additional bits used to make the condition codes for each instruction. These codes have reduced the chances of address since the processor does not have many bits to specify them but the great advantage is that these codes allow you to avoid jumps in the case of simple 'if' instruction.

Another unique feature about instruction set is the ability to shift data during normal data operations (arithmetic, logical, and copy records).

These features make the ARM programs normally denser than equivalent programs for other RISC processors. Additionally, the processor uses less memory accesses and manages to better fill the pipeline.

An ARM processor also has features which are rarely seen in the RISC processor such as the address on the PC (the Program Counter, PC, is a 16-bits register in ARM), addressing pre-and post-increment.

The first ARM processors like the ARM7 consumer were based on a design with a 3-stage pipeline: fetch, decode and execute. The most modern processors such as ARM11 has a 5-stage pipeline.

Other changes to improve performance include a faster adder and a system of branch prediction. In addition to native 32-bits ARM, there are two other interesting parts: the Thumb and Jazelle.

- Thumb code: instructions of 16 bits, lighter than the classic ARM but has less functionality. There are also two variants of the code Thumb: Thumb-2 which contains special instructions in 32-bits and Thumb-2EE specifically designed to manage codes for real time applications. The first processor with Thumb was the ARM7TDMI. All ARM9 and later families (including the XScale) are equipped with the Thumb.
- Jazelle code: it allows the processor to run natively on the Java byte code. This technology is fully compatible with the standard ARM and Thumb code. The first processor with Jazelle was the ARM926J-S, used on mobile phones to speed up the execution of software and Java games. The Jazelle code increases to over 95 % of the performance of the java code java, supporting 140 instructions and emulating with ARM the remaining 94 routines.

3.8 DSP Microprocessor

Digital processing is to represent the analog signal with a sequence of discrete numbers; processing with these numbers you can select the signal datas. The family of electronic components that perform these operations are well-known as DSP (Digital Signal Processors) [1, 2, 6–8].

DSP processors (Fig. 3.6) are microprocessors designed for digital signal processing that are applied in the field of wireless communications, audio and video processing and industrial control. A DSP processor is a high-speed dedicated integrated circuit specifically designed to perform arithmetic operations on large volumes of numerical data. It is similar, in physical structure, to a RISC processor, which recognizes only a small number of instructions, but these instructions are running very quickly. Many DSP chips are also included the specialized peripherals, for example, I/O circuits and timing of high speed memory, which typically include fixed-point processor or unit (depending on the mode of handling/storing of data). The fixed-point mode is designed to handle positive or negative integers; the other mode, however, guarantees the manipulation of rational numbers with 32-bits (word) available. The architecture includes an address bus with addressing mode specialized for the support of the signals operations. An important feature is its low power consumption and power management systems support the computation-intensive chips.

In DSP processors are integrated on a single chip functions of a typical computer. Obviously, this will help to choose an appropriate choice of memory. The main

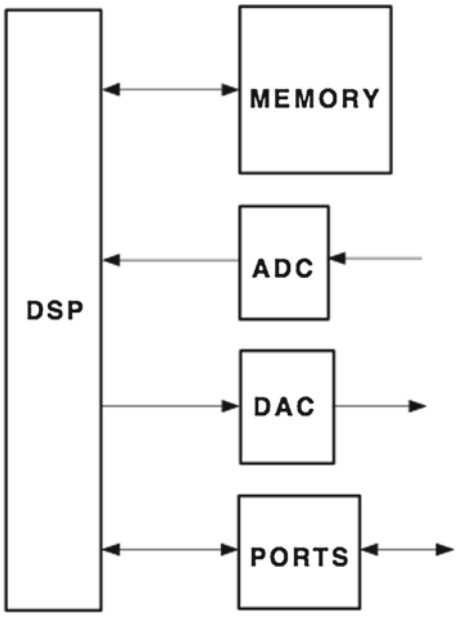


Fig. 3.6 General architecture of a DSP

operation is for the analog-digital converter (ADC) and digital-to-analog converter (DAC).

A key feature of the DSP, besides the speed, is that the running time must be predictable; it is necessary to have a precised calculation of the response time in the application, the rate of the response shall be in agreement (and constant) with the application, too.

Frequently, the DSP Algorithm contains instructions that must be repeated, instead of updating a counter or to execute a jump instruction, the DSP can solve this problem by hardware. There are in fact hardware structures that perform the control loops (Single and Multi-Instruction Hardware Loops).

3.8.1 Evaluation Parameters for a DSP

A processor may work well for some applications, but it is a poor choice for others. Selected a DSP, there are a number of characteristics from one to another to analyze:

- Format arithmetic: one of the key features of a DSP is the arithmetic type of format in use. The majority use the fixed-point, some are using the floating point where the numbers are represented by a mantissa and an exponent. The floating structure is more flexible and designers can access into a wide dynamic range. As a result, floating point DSP is generally easier than those fixed point but are usually more expensive and have a higher consumption of energy.
- Bus date: all DSP floating point data using 32-bits/word. For fixed-point DSP, the most common size is in 16 bits/word. The data size has a strong impact on the cost, because it strongly influences the chip size and the pin numbers required for the packages, as well as the size of the external memory devices connected to the DSP. The family of the Analog Devices ADSP-21xx, for example, uses a 16-bits data word and 24-bits word instructions.
- Speed: a key measurement of the suitability of a processor for a specialized application is its execution speed of the instructions. There are various ways to measure the speed of a processor; the most fundamental way is to calculate the cycle time of the processor instructions: the required time to perform the fastest possible instruction on the processor.
- Memory: a processor memory can have a big impact on its performance. Most DSP processors share some common basic features designed to support high-performance and numerically intensive tasks. The most cited of these features is the ability to perform one or more multiply-accumulate (often called “MAC”) operations in a single instruction cycle. Running MAC requires the retrieval of a word of instruction and two data words from memory to an effective rate of every cycle instruction which there are various ways to achieve, including by using Harvard architecture (Fig. 3.7).
- Power Management: DSPs are used in portable applications such as mobile phones, where the energy consumption is one of the main issue of the designer. Many

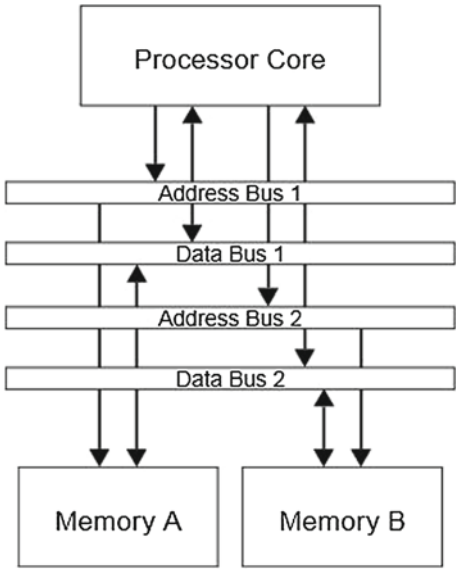


Fig. 3.7 Architecture Harvard

DSP designers introduced additional functionality for power management of the device, providing programmers a greater influence on the processor in terms of power consumption. Examples of energy-saving techniques on DSP are listed in the following points:

– Low Voltage Operating: consumption of about 5 times less at the same clock;
– Mode 'sleep' or 'idle': suspension of the device when not in use;
– Programmable clock divider: change the clock speed;
– Peripheral control: deactivation of peripheral devices not in use.

3.8.2 Commercial DSP

The world's largest manufactures of DSP are: Texas Instruments, Analog Devices, Motorola and ST Microelectronics. The C6000 DSP platform from Texas Instruments includes the fixed-point and floating with the highest performance in its class. The SMJ320C67x DSP are floating point processors for high-performance applications. Developed by Texas Instrument, the DSP (VLIW architecture) is an excellent choice for multichannel and multifunction performance with up to 1 billion operations per second (GFLOPS). Include an on-chip memory and a powerful set of programmable devices.

Very Long Instruction Word or VLIW refers to a processor architecture designed to exploit instruction-level parallelism (ILP). While the conventional processors allow

only programs that specify the instructions to execute one after the other, a VLIW processor allows programs explicitly to run the statements at the same time (i.e. in parallel). This type of processor architecture is intended to allow higher performance without the inherent complexity of other approaches.

Examples of CPU VLIW media processors are the NXP (formerly Philips Semiconductors), the Analog Devices SHARC DSP, the Texas Instruments C6000 and family ST200 STMicroelectronics. These CPUs are mainly used as embedded multimedia processors for consumer electronics devices.

3.9 Microcontroller as Embedded System

Today's approach is to pull over to the usual logical program of a microcontroller or a DSP, and then to realize a compact system with everything in needed to design a wide range of systems. Such devices are called SoPC (System on a Programmable Chip).

The fact to have a microprocessor and a memory integrated represents a decisive step in electronic design, without placing the additional discrete components, and the opportunity to realize truly integrated systems at low cost.

The embedded CPU in Programmable Logic are currently classified into six categories: ARM (ARM922T Altera, ARM7TDMI for Triscend) , PowerPC 405d (Xilinx), MIPS (Quick-Logic), 8051 (Triscend, Sidsa, Cygnal), AVR RISC (Atmel), M8C (Cypress MicroSystems).

The integration of microcontrollers in the FPGA enables the realization of efficient real-time systems that can respond immediately when unexpected events occur, especially if you use simple FPGA that have strong timing issues (clock skew).

Introduce a microprocessor capable of operation at specific frequencies, allows realizing reliable systems in a easy way. The need for real-time systems is the ability to have fast processors and on-chip embedded memory, suggest the use of real operating systems to manage everything on the device (Kernel). Several companies have already turned in this direction by promoting their products and realizing kernel owners for their devices. It's worth to note FREERTOS project, which allows free download from the site to install on your microcontroller into a real operating system. The performance gap between the microcontroller and microprocessors is gradually thinning out and it is reasonable to assume that within a few years this gap will be only a story [1, 8].

3.10 FPGA

Field Programmable Gate Arrays (Fig. 3.8) are two dimensional arrays of logic blocks and flip-flops with electrically programmable interconnections between logic blocks.

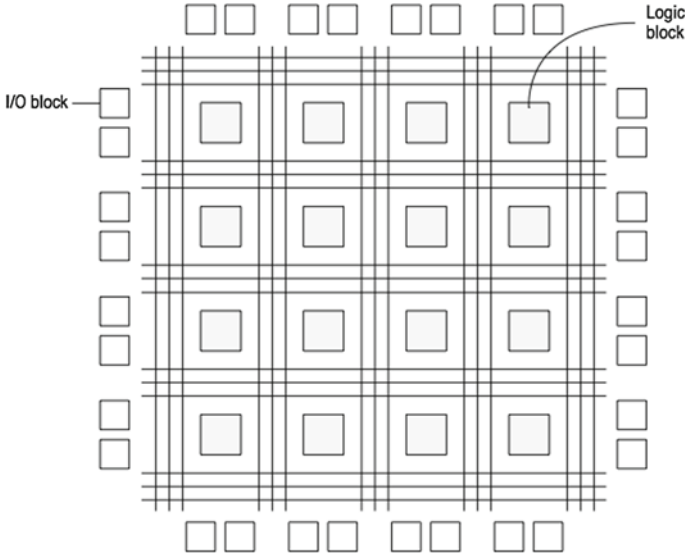


Fig. 3.8 FPGA structure

The interconnections consist of electrical programmable switches which is the reason that why FPGA differs from Custom ICs, as Custom IC is programmed using integrated circuit fabrication technology to form metal interconnections between logic blocks.

In an FPGA logic blocks are implemented using multiple level in gates, which gives it a more compact design compared to an implementation with two-level AND-OR logic. FPGA provides its user to configure:

- The intersection between the logic blocks;
- The function of each logic block.

Logic block of a FPGA can be configured in such a way that it can provide functionality as simple as that of transistor or as complex as that of a microprocessor. It can be used to implement different combinations and sequential logic functions. Logic blocks of a FPGA can be implemented by any of the following ways [7, 8]:

- Transistor pairs;
- Combinational gates like basic NAND gates or XOR gates;
- N-input Lookup tables;
- Multiplexers;
- Wide fanin And-OR structure.

FPGAs have been introduced as an alternative solution to customized ICs for implementing entire system in one chip and provide flexibility to re-programme by the users (Fig. 3.9).

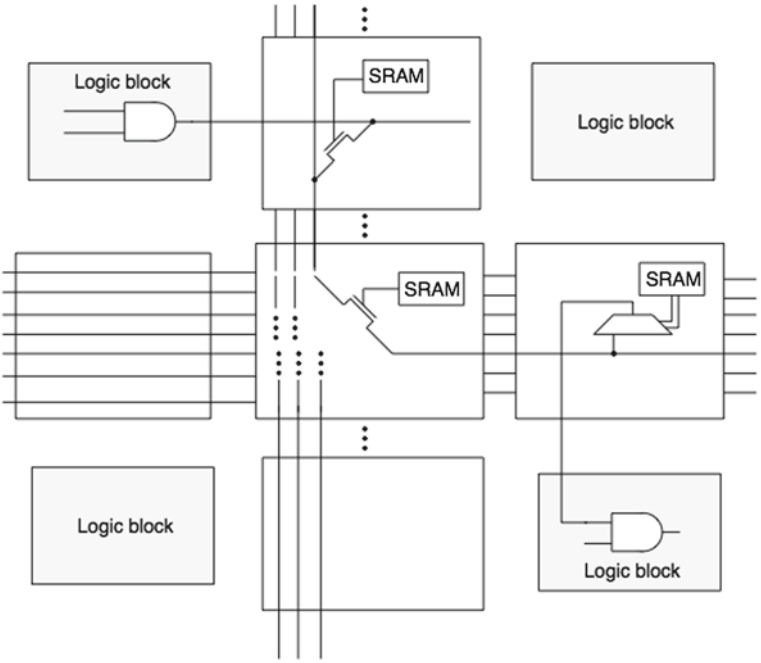


Fig. 3.9 SRAM-controlled programmable switches

References

1. Park, J., & Mackay, S. (2003). *Pratical data acquisition for instrumentation and system control*. Boston: Elsevier.
2. Razavi, B. (2008). *Fundamentals of microelectronics*. New York: Wiley.
3. Kang, S. M., & Leblebici, Y. (1998). *CMOS Digital integrated circuits*. New York: McGrawHill.
4. Heath, S. (1997). *Embedded systems design*. Oxford: Newnes.
5. Vahid, F., & Givargis, T. (2002). *Embedded system design: A unified hardware/software introduction*. New York: Wiley.
6. Wilmshurst, T. (2008). *Designing embedded systems with PIC microcontrollers*. Oxford: Newnes.
7. Ganssle, J. (2008). *The art of designing embedded systems*. Oxford: Newnes.
8. Valdes-Perez, F. E., & Pallas-Areny, R. (2009). Microncontrollers, fundamentals and applications with PIC CRC Press.

Chapter 4
Design Techniques of Embedded System

Abstract Embedded systems (ES) are a special-purpose computer system designed to perform one or few dedicated functions. The development of embedded system is getting difficult due to the short life-cycle of the embedded products and high complexity of embedded systems design. In this chapter, we analyze a possible method to design Embedded Systems.

4.1 Design

The major computers used in the industrial field are embedded systems which are found and placed in mobile phones, mp3 players, cameras, cars, appliances. These devices usually have a static single function, unlimited, repeated, with tight constraints of price, power and performance. Moreover, it must react to change in an instant response without slowing down (real time). Over the years, embedded systems are becoming more and more powerful despite that their size continues to decrease, thanks to the progression in electronics. Initially, these devices were programmed entirely in assembly language in order to optimize the code and reduce power consumption, but now are used high level languages and sometimes it makes use of middleware applications to simplify and speed up the process of software development.

The role of design is important both from hardware and software point of view, two aspects that we will analyze in parallel by using techniques of co-design as a media representation languages that define the system at a high level hardware (VHDL, SystemC). Depending on the application (server PC, Telephony and so on) that can be desktop or embedded systems, they have a different implementation (design style, technology ...) and then are programmed properly at both HW (reconfigurability) and SW (at the application and education level) level. In the case of embedded digital systems, it is necessary to determine the reactivity of the system (must respond in micro/milliseconds), the hardware and software on which to place the features (FPGA, ASIC, operating systems, drivers) to improve the performance by reducing the cost and power consumption. Through certain types of design flow, it is possible to analyze the problem of the realization of a digital system, reducing the time

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_4

of analysis and implementation (usually using semi-automatic mechanisms) and optimizing the final system. A new approach is that the embedded systems-on-chip which consist in the inclusion of various components (DSP, memory, drivers and so on) is included in a single integrated circuit. The evolution of system-on-chip opened new scenarios with more importance to the language specification and modeling of the system level, enabling the high-level design and then take advantage of automatic and pre-designed algorithms to speed up and optimize the development of embedded system. The open problems about the complexity of embedded systems can be the following:

- To understand the behavior and design of a complex systems.
- How to translate the implementation specific?
- How to check the realtime constraints and power dissipation?
- How to test the systems before the implementation?

The design flow may be partially or fully automated, using the tools such as compilers, software engineering programs, starting from the highest level of abstraction to the low level.

The design flow used in recent years is the HW-SW Codesign: a design that emphasizes the unitary vision between the hardware and software parts, exploiting the various features in a development parallel avoiding, in this way, the design errors.

The main steps of the Codesign are:

- Requirements analysis: To analyze functional (to describe output as a function of the inputs, i.e. to define the functionality of the system) and non-functional (performance, space, weight and so on) constraints.
- Co-specific: modeling the level of system with formal languages (i.e. C, C++) which allows to represent both the software (SW) and Hardware (HW).
- Management of concurrent tasks: it defines the granularity of parallel processes and their management.
- High-level transformations: to elaborate high-level optimizations ("roll out" of the loop).
- Space Exploration Project: to evaluate different designs.
- Co-synthesis: To get automatically the implementation of the system: hardware (dedicated units), Software (processors), user interface and synchronization. It consists of partitioning and scheduling phases: Partitioning phase chooses the various components and check them in SW or HW section. Two approaches are possible: the HW-oriented partitioning minimizes the costs by creating initial models in hardware (VHDL) and then brings the non-critical parts in the software. The oriented software approach maximizes the performance with initial software (C++), which then elaborate the critical parts in hardware.
  Scheduling: it identifies a relationship between time and the running process, choosing one of the many different approaches depending on the scope of the project.
- Co-simulation: To check if your system meets the initial specifications without errors. To simulate the hardware and software sections, check and validate the consistency integration between the parties.

The use of this type of design flow allows to promptly identify errors, sections where reuse previously written code and perform appropriate tests trying to reach for a compromises in performance/cost. All this helps for a better and quick design of systems, providing huge benefits to producers who have narrow windows time-to-market to be respected [1–3].

4.2 The Waterfall Model

The waterfall model (Fig. 4.1) consists of six main steps:

- Feasibility study;
- Analysis and specification of requirements;
- Project;
- Architectural design: it defines the general structure of the system;
- Project in detail: it defines the modules individually;
- Coding: the parties are implemented and integrated with the hardware;
- Testing;
- Maintenance.

The development phases are organized in a strict hierarchy, in according to a design level where there will be necessary to define the next higher level. It would be very expensive to make changes to a higher level during the hierarchy selection, while the definition of software components for each module will be necessary to determine, for which functionality should be implemented in hardware and software.

The hardware/software portioning is a complex problem of optimization, where must take into account of several factors (cost, competitiveness of the product, performance, proprietary hardware) which they often conflict with each other.

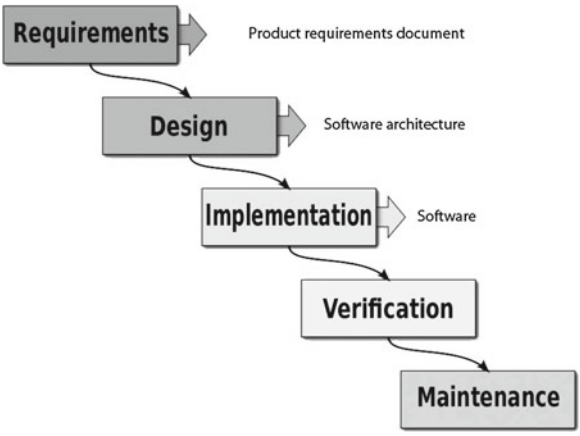


Fig. 4.1 Waterfall model

Just about the competitiveness of the product may hinder the achievement of a fully satisfactory solution. Rather, it will reach to a product with necessary compromise to satisfy the goals of client. The phase of elicitation and analysis of requirements will determine an appropriate and essential hardware/software partitioning, after then, it will define the general architecture of the embedded system and, of course, the interface between the two logics. Downstream of what, it will be necessary to determine the specifics hardware and software. The advantage of this method is, the possibility to produce separately and simultaneously hardware and software components. In addition, it is important to understand which hardware or software modules must to be built from scratch and which, instead, can be reusable or purchased externally. Note that usually embedded systems are developed by a reference platform. It is a device, or a basic architecture, easily modifiable in order to make changes to existing functionality or to build new ones. This strategy becomes especially useful if the system is realized with dedicated and defined functions regulated by the standards. This development approach consists of two phases: the first, is defined in the platform (base functionality, performance, and reconfigurability) and the second, the platform is used to design a specific product based on it. In evolutionary models, it arises the goal to overcome the limits of the waterfall model, in order to obtain a greater flexibility.

Belongs to this category, there is the spiral model (Bohem 1988), where the development is interpreted as an incremental process consisting of small consecutive steps. It combines the iterative nature of prototyping, the control and systematic linear sequential model.

About the software development, this method requires that each cycle consists of all stages (which can be summarized in requirements, architecture and coding), and that each of them gives an enlarged and detailed version of the product, since each cycle appropriates the knowledge and experience gained in previous cycles.

The version obtained at the end of each cycle can be a project, a prototype or a software product.

The spiral methodology consists of several steps that are repeated cyclically:

- Communication with the customer;
- Planning;
- Analysis of risks;
- Structuring;
- Construction and release;
- Evaluation by the customer.

Unlike the rigid waterfall model, the spiral model will allow making modifications and refinements in each phase with a greatly reduction in cost. This approach, however, is particularly risky where the process should be strictly controlled; In fact, in the spiral model would be much expensive to produce all documentation in order to support each activity performed [1–3].

4.3 Model V

Where the reliability, safety and quality are imperative requirements, such as occurs in embedded systems, the typically development model used is the waterfall.

Despite the limitations, it guarantees the execution of a well-structured activies. Moreover, it allows carring out projects which products are constructed, at least theoretically, in a sequential manner.

The waterfall model is based on the following two assumptions:

- Users requirements are clear and the technology is familiar for developers and client;
- The client accepts that the last time step of the project is related to the function of all output produced.

The V-model (Fig. 4.2) was born from the waterfall model and is characterized by the following steps:

- By non-local interactions between the different phases;
- The presence of recycle, not only relate to the previous stage;
- The presence of a critical phase, review of the entire project.

Typically, the V-model is characterized by 9 stages of development 'symmetrical', in each step of the process will need to draw up a formal document; the output of the entire project, which will have to be approved by the responsible personnel.

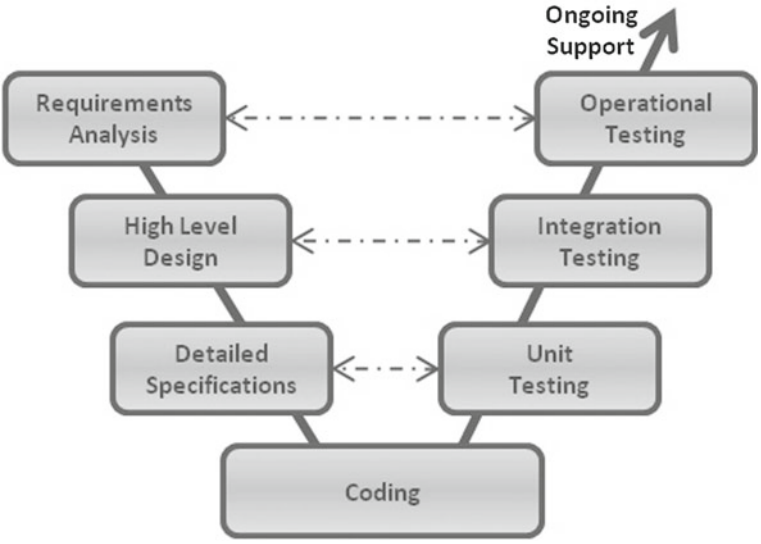


Fig. 4.2 Model V

The phases of the V-model are the following:

1. pre-analysis/feasibility study: The goal of this phase is determined the economic opportunities of the project, in terms of cost and benefits. Therefore, it will also consist of the examination of possible alternative embodiments and in choosing a solution among them. The decisions made at this stage of the project will not be definitive, since the user requirements, as well as technology, are not yet fully known. During the feasibility study, as well as in three successive stages, should make decisions that are as much as possible independent of each other.
2. User requirements elicitation and analysis: these two activities are relatively unstable, since this point of the project and the client is unlikely to be able to express completely their requests. On the basis of this consideration, and that the fundamental formal document drawn up at the end of the activity is 'modular'. In practice, it will have to be synthetic (in the right size) and will have to be readable for developers and users, while not yet clear requirements and objectives, will approve the content.
3. System design: in this phase the document is completed, which contains the detailed description of hardware and software architecture for the whole system.
4. Program design: on the basis of the functionality described in the process of elicitation, analysis and specifications technology defined during the system design, it will be documented in detail when all the software modules are implemented. They will be reported in a formal document, which represents the output of this phase.
5. Programming/Coding: at this stage the programs and the necessary documentation for their maintenance are produced; while, in contrast to what happens in the waterfall model, user manuals are still yet to be finished.
6. Program test: at this stage the database of test are produced and populated in order to verify the functionality and performance of each module completed. At this point, for each module accepted, will be drafted in the user dumentation.
7. System testing: testing of all programs, including directors and run of a test on the overall performance of the system. It will have to ensure that the overall performance is acceptable and that the macro (such as the regulation of access, security and so on) are properly designed. At the end of the stage, it will be produced the necessary documentation for the installation and operation of the system.
8. Acceptance by the user: this step is not explicitly included in the waterfall model. During this activity, it will be necessary to check all the available documentation.
9. Review and start-up project: in this last phase the installation of hardware, software, will be completed , generated and populated the database management. The training of the end users of the product is required.

The commission is a long activity, which may also end with the cancellation of the entire project, if it is not really possible to use it. In the latter case, through a careful and well-documented project review, all the work will constitute a solid base of knowledge and expertise.

The most effective application of the V-model is related to the case in which a company wants to proceed with the first computerization of a given area, for which there has been sufficient experience arising from successfully completed projects [1–3].

4.4 Architecture

The architecture of an embedded system and its technological aspects are determined by the processor (SW) or chips (HW) and design constraints due to the scope and the required flexibility. The processor used in an embedded system can be in different types depending on the functionality required:

- The general purpose processors (GPP) are flexible and inexpensive programmable devices which contain a memory program and datapath unskilled to perform most operations of various kinds: Load (load from memory to registers), Store (except from registers to memory) and all ALU operations. Examples of GPP are microcontrollers, RISC processors, DSPs and multimedia processors. An optimization of this architecture is through the introduction of the concept of pipelining, a technique that allows the simultaneous execution of more instructions in the style of an assembly line. Latency remains the same, but the speed of execution of more instructions is increased. It is possible to increase the performance by using static or dynamic scheduling (VLIW), or using superscalar processor.
- The single-purpose processors (SPP) digital circuits are designed to perform just one type of program. This category of processors doesn't contain a memory program and the datapath has a simple logic suited to perform only a small and static set of instructions.
- Processors for specify application are programmable processors optimized or a class of applications with common characteristics. They are a compromise between the two previous models with a memory program and a special datapath optimized with functional units (ASIP).

A very common ASIP microcontroller is used to manipulate small amounts of data in embedded systems and contains data memory and programmable on-chip peripherals via pins. Another ASIP is the Digital Signal Processor (DSP), which is used in the applications that process large amounts of data stream quickly. DSPs have many functional units and ALU buffer vector. An integrated circuit (IC) consists of over transistors, such as CMOS and PMOS.

There are three different integrated circuits technologies depending on the level of customization of the different layers: Full-custom, Semi-custom and Programmable Logic Device (PLD).

Full-custom: the IC (VLSI) are fully customizable, so is possible to have great performance, small spaces and low power consumption, but with very high NRE costs. It is necessary to establish the position and orientation of the transistor, their connections and so on. Circuits of this category are manufactured by Intel.

Semi-custom circuits (ASIC) have lower pre-programmed levels, then start from a general structure which is then adapted to the particular digital functionality. NRE and unit costs have acceptable performance and good power consumption. They can be gate array (an array of prefabricated gate) or Standar Cell.

PLDs are less expensive and simple, provide complex logical components that can be connected to each other depending on the goal of the project. PLDs have high power consumption but lower NRE costs and time-to-market.

The choice of the architecture and metrics related to the project is also based on the type of system in request, which can be 'dominated by control' or 'dominated by data' (they have a steady stream of data, require buffer memories and DSP), although generally the ES are mixed systems [2, 4–9].

4.4.1 ASIC-ASSP

ASICs and ASSPs have been able to develop, in a short time, a real stronghold in the market of electronic components. ASSPs are manufactured to be sold to any person who requests it. With this, they are different from ASIC (Application Specific Integrated Circuit), which are exclusively designed to the customer. Both types have the same characteristics in terms of size, performance and miniaturization. Smartphones, laptops and other multifunction devices represent a significant application-specific standard product (ASSP) and Application Specific Integrated Circuit (ASIC). The market dell'ASSP and ASIC is the largest and only for semiconductor test equipment (STE).

In electronics, an Application Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general use; for example, a circuit designed to operate in a digital recorder is an ASIC. Standard products for specific applications (ASSP), instead, are intermediate between ASIC and standard integrated circuits, such as the 7,400 or 4,000 series. Within ASIC chip can integrate various components, such as bipolar transistors up to 80 V, NMOS, CMOS logic, JFET and various sensors such as hall effect. The ASIC may arrive to integrate a very large number (millions) of logic gates: integrated circuits of this type are called System on Chip (SoC) and are programmed with appropriate programming languages such as VHDL. The chips designed for specific applications represent an evolution of the market that benefits the embedded devices development, where the personalization of functionality is required.

The ASIC design (Fig. 4.3) must be in accordance with the client and the best solution to the problem. This will occur if there are no other solutions on the market and at an affordable cost. The aspects to be followed to assess are the factors of cost and design time and, then, the implementation of the integrated circuit. First step is to evaluate the design specifications of the system to achieve. The design of an ASIC is obviously done with CAD software that simplify the design and testing of the functional circuit simulation by tools.

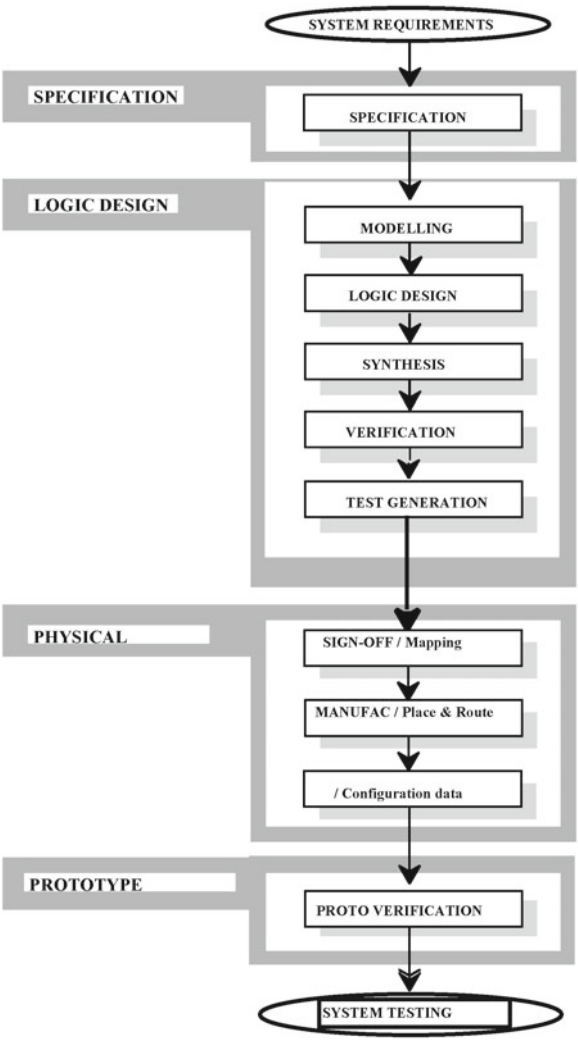


Fig. 4.3 Flow of ASIC design

At the end of the software simulations the circuit that is located in a descriptive phase at the level of the gate (gate -level), should be mapped in an integrated circuit: the way in which this occurs differs from industry to industry.

From a point of view of design, the ASIC can follow two types of approach: Full Custom or Semi Custom as described previously. The Full Custom consists in the design of the integrated circuit at a low level (for example, the design of a single cell transistor) drawing masks, and extend all at higher levels (for example, combinations

of cells for higher-level functions) to create the overall function of the IC. In Fig. 4.4 it is shown the general pattern at the cell level.

About the Semi-Custom, the design level is higher by using circuit blocks under existing or planned. In addition, in the design of custom style, each function or primitive logic transistor is designed and optimized manually. This involves the design of more compact IC with the highest speed and lower power dissipation. Semi Custom circuit design can be of two types: array devices and standard-cell- or cell-library-based components. The category is the use of Cell-based, cells already predefined in special libraries, called Standard Cell; or the automatic synthesis design of macrocells from their logic (Cell Generators).

Array-based design is proposed to reduce the time of realization of the ASIC and the costs of the design; this is realized on the basis of a generic integrated circuit, which is then customized by the designer. The Array-based can be divided into: Prediffused and Prewired. In the first case within the gate array (Fig. 4.4b), Compacted Array and Sea of Gates. In the second case, however, fall within the FPGA (Fig. 4.5), i.e. those programmable field.

The design flow of an ASIC is the sequence of steps to follow to achieve a integrated. Below is listed the protocol implementation:

- Design Entry: design through a specific language or by HDL or VHDL schematic description;
- Logic Synthesis: creating a description of the logic cells used and their connections;
- System partitioning: the ASIC division into sub-blocks;
- Simulation of Pre-Layout: first simulation to analyze the functioning of the project in according to the initial requirements. This simulation is only a logical verification which does not consider the possible delays of signals that exist in the logic gates;
- Flooplanning: positioning of the various blocks of the chip netlist;
- Placement: placement of logic cells;
- Routing: connections between cells and blocks;

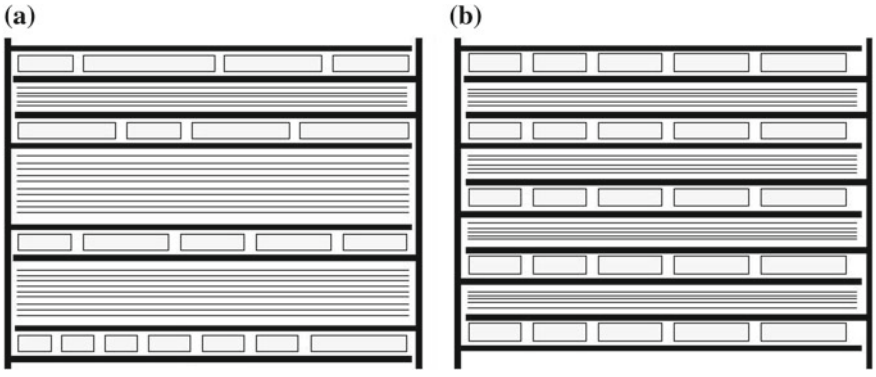


Fig. 4.4 **a** Custom layout, **b** gate array layout

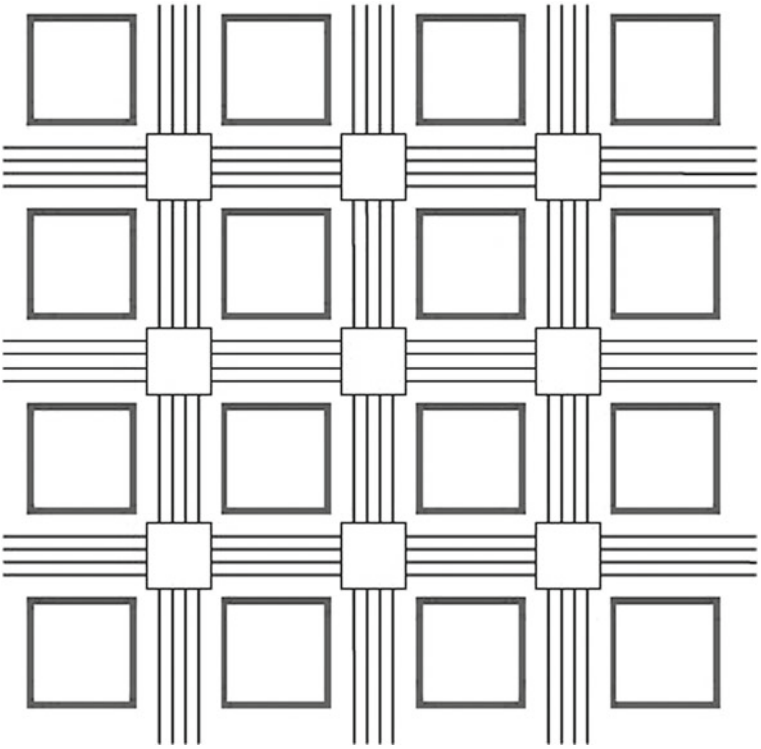


Fig. 4.5 Layout of an FPGA

- Extraction: calculation of resistance and capacity to assess the propagation delays;
- PostLayout Simulation: simulation and analysis of further delays.

The market for integrated circuits requires the use of the most advanced technologies to maintain high performance, to reduce costs to stay competitive in the market and in the product time-to-market.

LSI Corporation is the developer of semiconductors for excellence by providing standard integrated circuits (IC) and custom-designed application-specific integrated (ASIC, Fig. 4.6), focusing on broadband communications, wireless, data storage, personal computers, networks and markets. LSI (an acronym for Large-Scale Integration) was a pioneer of system-on-a-chip (SoC) devices that combine elements of an electronic system, essentially a microprocessor, on a single chip. Top customers include IBM, Seagate and Hewlett-Packard. LSI also provides hardware and software for storage area networks [2, 4–9].

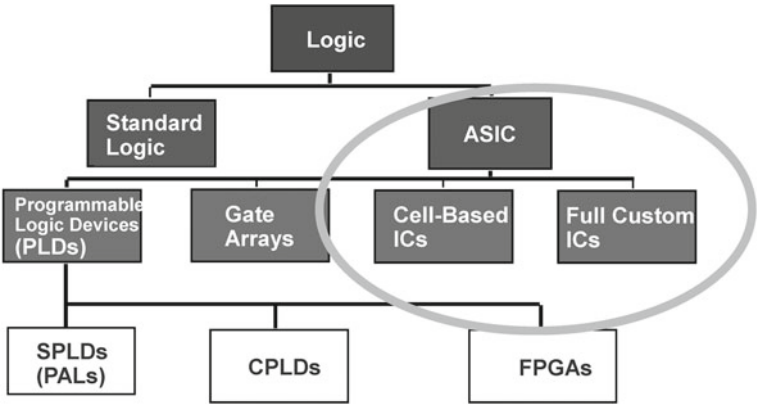


Fig. 4.6 ASIC design

4.5 Software

Embedded software is written to control machines or devices that are not typically work of the computers. It is basically specialized for the certain hardware with time and memory constraints. This term is sometimes used interchangeably with firmware, although firmware can also be applied to ROM-based code on a computer, on top of which the OS runs, whereas embedded software is typically the only software on the device in question.

Unlike standard computers that generally use a small number of operating systems (largely MacOS, Windows and GNU/Linux), embedded software comes in a wide variety, typically real-time operating systems. Example are the following: LynxOS, VxWorks, BeRTOS, ThreadX, to Windows CE or Linux (with patched kernel). Others include OpenWrt, PikeOS, eCos, Fusion RTOS, Nucleus RTOS, RTEMS, INTEGRITY, uC/OS, QNX and OSE. Code is typically written in C or C++.

Hewlett- Packard has developed a set of quality attributes of the software which has been assigned the symbol FURPS (functionality, usability, reliability, performance and supportability). These attributes represent a target for all software projects:

- Functionality is assessed by checking the capacity of the program, the generality of the derived functions and safety of the overall system.
- Usability is assessed by considering human factors, consistency and documentation.
- Reliability is assessed by measuring the frequency and severity of failures, the accuracy of output results, the ability to recover a fault condition and the predictability of the program.
- Performance is measured in terms of processing speed, response time, resource consumption, productivity and efficiency.

- Supportability combines the ability to extend the program, adaptability, ease of service, testability, compatibility and configurability.

The software development process describes the activities (or tasks) necessary for the realization of a software product and how these activities are linked together. Software engineers adapted the process to their goals and presented the important parameters of stability, control and organization in an activity:

– A software process is a set of activities and their results allow to produce a software system.
– A software model is a simplified description of a software process under a particular perspective.

The software development process (Fig. 4.7) can be defined as a collection of phases which define a set of activities, actions, work tasks to help in the realization of the software. In general, a software model provides a structure, a consistent outline to describe an important feature of the process of software development. Combining these models, a software development team can create a process that responds better to the goals of the project.

Embedded-software systems pose extraordinary challenges to the software engineer due to their complexity. The main features can be described in the following texts:

- Functionality represented by states and events;
- Real-time behavior of events and expected actions;
- Combined software/hardware systems equipped with distributed software, computers, sensors, and actuators;
- Long-lived systems in which embedded software is expected to work reliably.

Common denominator across all embedded-software domains is the use of programming languages that allow direct access to interfaces, memory and so on. More than 80 % of all companies are using C and C++ language.

Most of the methods for software development consist in a modeling language and a process:

– the modeling language is the notation used by the methods to express the characteristics of the project;
– the process is the list of information regarding the steps to produce the project.

The UML (Unified Modeling Language), for example, is a modeling language used by processes to implement, organize, document the products manufactured by the steps of the process which it is composed.

Working on the software development, each individual or team, will necessary adopt a certain approach in the way related to their customers / users organizing their work in the choice of techniques to be used.

Consciously or not, any developer (or team of developers) has its own software development process—a way of working, based on his own experience, the culture and organizational context of work.

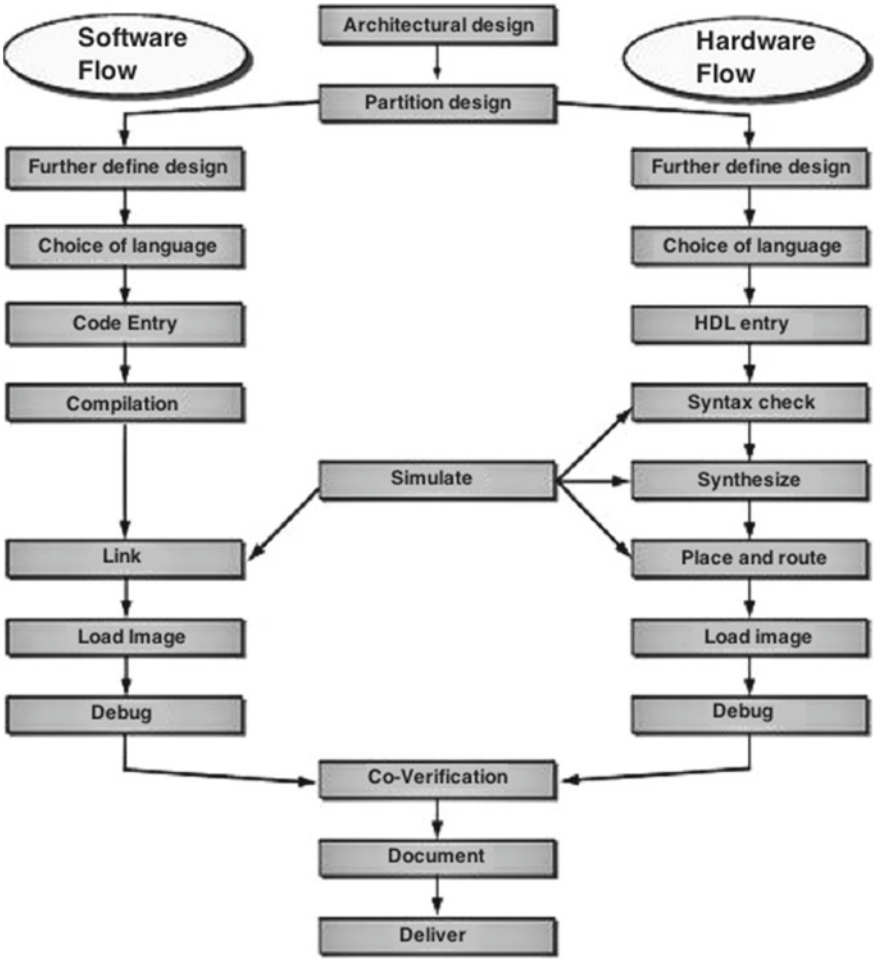


Fig. 4.7 Software design

In most cases, the life-cycle of the software development is iterative and each iteration produces a its release. Below the main steps that can be part of each iteration:

- Specification of requirements: requests by the customer;
- Feasibility study (make or buy);
- Analysis and design: analysis is the study of what the system must do ('logical' point of view) and the design is the study of how the system should be implemented ('technician' point of view);
- Implementation;
- Integration and testing.

If the original models of the life-cycle software tools were purely conceptual, modern development for environments, CASE (computer aided software engineering), usually automate the application of a particular life-cycle as well as other aspects of a software process.

Some examples of models are the following:

- The waterfall model involves the sequential execution of the phases of analysis, design, development, testing and maintenance.
- Iterative models: all development processes that allow to return to a stage of the proceedings already addressed.
- Models evolutionary: the waterfall model is no longer suitable to be applied. Among other factors that have contributed to the decline of this model are the increasing complexity of applications, the introduction of the GUI and the advent of new programming paradigms, methodologies of analysis and design, particularly those related to the paradigm of object-oriented.

4.6 Embedded Linux, Windows, Android

Embedded operating system, as real-time, is used to control the specialized hardware and it must meet timing and functional constraints. How to choose an operating system for embedded devices?

An embedded system may be thought of as the heart of a large electronic system. The complete set of components (microprocessors, DSPs, RAM, etc.) are controlled by a highly specialized embedded operating system.

The main operating systems on the market are the following: Windows, Linux and Android.

All these systems provide connectivity services and advanced support for many protocols. Linux systems can exploit for the embedded server components, unlike Android where the connectivity is oriented to the role of client.

Windows Embedded Compact provides a minimal support for server, functionality and components that enable the integration into network architectures. The software Microsoft is considered a hard real-time operating system with stringent timing constraints.

Windows Embedded Compact also provides tools for debugging kernel mode and debugging of applications. Its kernel codes are available in source format, as well as all the drivers, BSP and samples of different components of the system.

Both Windows CE and Linux support Web Services on Devices. The Linux kernel does not come as a real-time kernel, but there are patches to support real-time. An embedded Linux system can support applications and design services for the desktop or server.

The kernel is released under the GPL license and a multitude of services and applications for embedded systems are available in source format with various licenses.

The Android operating system, however, comes from the heart of Linux, supports cellular connectivity and a browser advanced, is developed and tested with a real-time

kernel; NDK provides for the development of applications and/or native components. The components of Android are released by google with a BSD-like license [2, 4–9].

4.6.1 Windows Embedded Compact

Windows Embedded operating systems includes Windows Embedded Compact for small devices such as smartphones, Windows Embedded Standard for the most advanced devices, Windows Embedded Enterprise for desktop device and Windows Embedded Automotive for automation systems.

Windows Embedded Compact (previously known as Windows Embedded CE or Windows CE) is the version of Windows Embedded for small computers and embedded systems, including devices for consumer electronics and gaming consoles. Windows Embedded Compact is a modular operating system in real time with a special kernel that can be performed in less than 1 MB of memory. It comes with the Platform Builder tool that can be used to add modules to the image installation, depending on the device used to create a custom installation. Windows Embedded Compact is available for ARM, MIPS, SuperH and x86 processor architectures.

Microsoft has also specialized version of Windows Embedded Compact, known as Windows Mobile, for use in mobile phones. It is a customized version of Windows Embedded Compact with specialized modules for use in mobile phones. Windows Mobile is available in four versions: Windows Mobile Classic (Pocket PC), Windows Mobile Standard (for smartphones), Windows Mobile Professional (for PDA / Pocket PC Phone Edition) and Windows Mobile for Automotive (for communication systems/entertainment/information used in cars) [2, 4–9].

4.6.2 Embedded Linux

Linux is a UNIX compatible, open source, originally designed for desktop computers as an operating system for a file server or gateway to access the Internet; Linux offers the main features and stability for operation without unattended reboots.

Linux is also widespread in the embedded systems market, it's a modular operating system that is available in source code appropriately documented and offers a multitasking features and advanced programming interface (API). Linux supports almost all 32-bit processor architectures, as well as some 32-bit microcontrollers, providing drivers for any hardware.

Basically an embedded Linux consists of three main modules: the bootloader, the Linux kernel and the root filesystem. There may be three basic elements in the form of a single binary image (4.8) or as separate files (4.9).

The simple and elegant design of Linux provides strength and very good performance, while its open source license allows to modify the source code in according to the needs of the users.

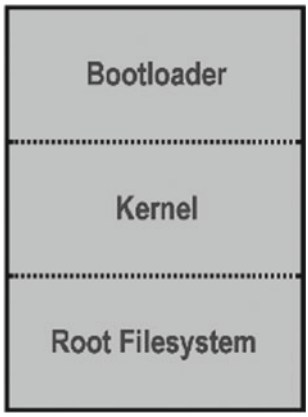


Fig. 4.8 Components of an embedded linux system (binary image)

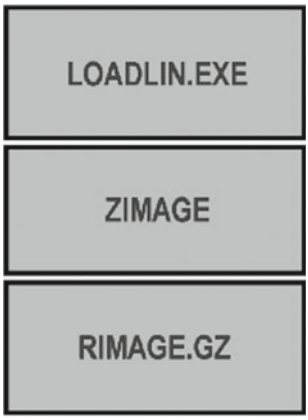


Fig. 4.9 Components of an embedded linux system (3 files)

By using Linux, you can get some important advantages:

- Availability of free source code and development tools;
- POSIX standard support to improve portability between different systems;
- Wide range of operating system services, including network support.

4.6.3 Embedded Android

Embedded systems control many devices that are in common use: they are ranged from portable devices such as digital watches and MP3 players, to large stationary installations, such as ATMs and vending machines. However, in recent years Embedded systems have changed dramatically; Nowadadys, they are largely multimedia, connected and highly integrated. Many graphical user interfaces, are included

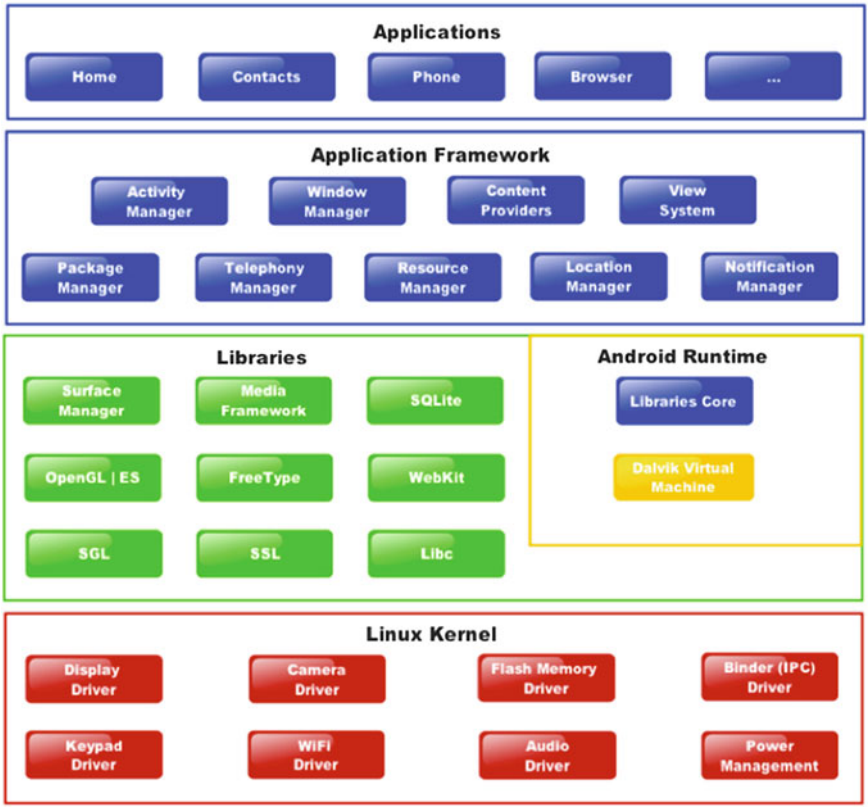


Fig. 4.10 Architecture of Android

high-resolution 2D and 3D graphics. In addition, almost all the embedded systems include stack IP networking and connectivity through a combination of wired and wireless network interfaces. And last of all, for reasons such as energy efficiency, size and performance, they are designed for minimal power consumption.

Android is open source and the majority of the source is licensed under the Apache 2 (4.10).

A system of Android is a stack of software components: at the bottom of the stack is Linux. This provides the basic functionality of the system such as process, memory management and security. In addition, the kernel handles networking and a wide range of drivers for interfacing with hardware devices.

At first glance, Android may be seem as an odd choice as an embedded operating system, but in fact Android is already an embedded operating system, its roots stem from the Embedded Linux. Android provides a user interface with multi-media operation, a stable kernel without royalties or licensing fees and an extensive library of source codes. All of these things combine to make the creation of an embedded system more accessible to developers and manufacturers.

Android has a large community of support from OEMs and SoC. Even if the support comes mainly from ARM, Android also supports x86 processors. This provides a wide range of hardware configurations to choose in according to your budget and system [2, 4–9].

4.7 Power Management

Dynamic management of power consumption in the context of embedded systems is a very important technical aspect in the design and will have a high increase in the future; it has several advantages including the improvement about the energy efficiency of the system.

The energy consumption of an electronic component in CMOS technology mainly comes from three contributions:

- Energy consumption is generated due to charging and discharging of the output capacitance;
- Reverse bias current;
- Short circuit current during switching.

In Fig. 4.11 is shown a simple MOS inverter circuit where the transition from high to low logic level of the input signal assumes a slew rate of finite value with closing times (PMOS) and opening (NMOS) simultaneously. There will be a certain time in which the two transistors are in the saturation zone and show a finite resistance: the energy dissipation will be proportional to V_{dd} and the power will scale as the product of the frequency for the V_{dd}.

The parameters on which to intervene, therefore, to propose a reduction of energy consumption are the frequency and the working voltage. Whereas most of the consumption of CMOS systems is also due to the switching of the circuits, it is useful to consider the possibility to avoid the clock trigger for components that are not involved in the switching.

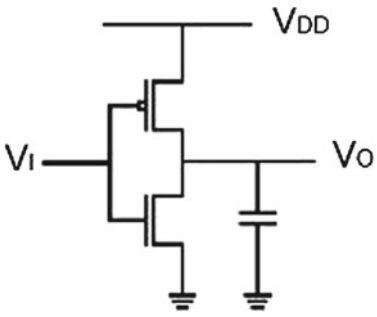


Fig. 4.11 MOS inverter circuit

In this case, units are required for the conduct of each step of education and to be sure that during their execution, the instruction decode unit enables the clock line addressed to the respective units: each clock is made to flow in an AND gate.

The technique of frequency, however, is an excellent strategy that can be made with some design features of the PLL which generates the clock.

A typical embedded system is constituted by a basic hardware platform that executes the software system and application connected to a number of devices. Examples of devices are on display, wireless local area network, keyboard and camera. The power efficiency is obtained by reducing the power dissipation in all its parts. Reduce the energy consumption of all the hardware constituents means to be able to minimize its power dissipation.

The implementation of the software (i.e. the kernel of the operating system, application-level code) affects the power efficiency of the basic hardware and related peripherals.

For example, the efficiency of compiled code and memory access patterns play an important role in determining the overall power dissipation of the embedded system [2, 4–9].

4.7.1 Dynamic Power Management

The dynamic power management (Dynamic Power Management, DPM), which refers to the selective closure of the system components that are inactive or underused, has proven to be particularly effective for reducing the power dissipation in embedded systems. It's a process that requires many iterations of design and debugging. The goal of the DPM is, therefore, reduce the energy consumption of an electronic system by placing the various components in different states, each with a level of performance and consumption. The strategy will determine the type and the transition timing of these levels on the basis of the history of the system and constraints.

The activity of many components in a computing system is event-driven. An intuitive way to reduce the average power dissipation, by the system is to turn off the resources during their periods of inactivity. In other words, one may adopt a energy-saving policy that dictates how and when the various components must be turned off.

A simple and widely technique is the 'time-out', which activates when the component is used and turns off when the component is not used for clock periods. The effectiveness of this technique depends largely on the timeout, as well as the characteristics of the component.

There are a number of problems which a designer must be aware at design time:

- Voltage and frequency of the operation: the operating system must be able to monitor the working voltage and determine when the frequency of operation should be changed.
- Processors can not be operated reliably during the transition phase of the voltage or frequency: many CPUs, capable of operating at different voltages and corre-

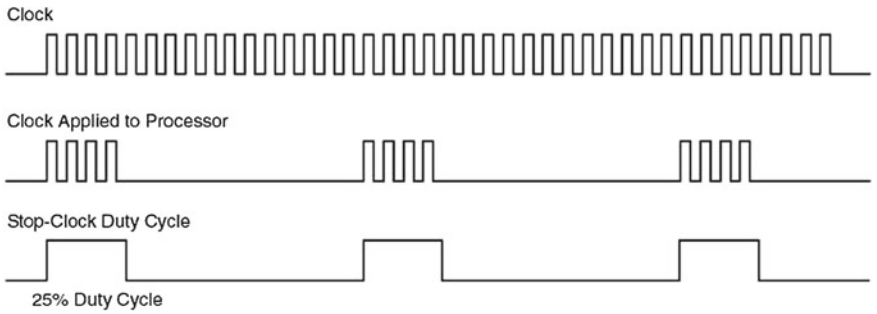


Fig. 4.12 Bursting clock

sponding frequencies, may not operate when their supply voltage or the frequency of the clock are changing. In such cases, it is advise to stop the CPU during the transition of the voltage and/or frequency. This requires an external hardware to monitor the voltage and clock frequency to prevent that the CPU is running during the transition.

- Usually, CPUs with integrated PLL generate the necessary frequency for the integrated peripherals and provide the clock for the external bus. If the clock frequency of the CPU has been changed, the PLL should be reprogrammed to maintain the frequency of operation for external devices that are not designed to operate at different frequencies. An external PLL can easily overcome this limitation and expand the range of energy saving.

Similar effects can be achieved without acting on the PLL (and therefore without having to suffer the latencies) by means of the bursting of the clock, which is also known as a stop-clock. This mechanism is realized by generating a second signal with controllable duty cycle by sending it to an AND with the clock, obtaining a signal (Fig. 4.12) used to drive the microprocessor.

4.7.2 Dynamic Voltage Scaling

Dynamic Voltage Scaling is a power management technique, where the voltage used in a component is increased (overvolting) or decreased (undervolting), depending with the circumstances of work. Undervolting is done in order to save energy, especially in laptops and other mobile devices, where the energy comes from a battery and is therefore limited. Overvolting is done, however, in order to increase your computer's performance, or in rare cases, to increase the reliability. By running the components at a low voltage, it is possible to reduce the energy consumption, but at the same time, it also reduce the performance of the system.

With this technique, it is possible to control the temperature of the processor, by decreasing the power consumption which is important in the cool phase.

The use of this technique should be done with particular care, because it increases the wear of the components, with consequent increasing of the execution time of the operations and decreasing trend in the service quality desired.

4.7.3 Latencies

The examined techniques induce problems of latencies of electrical transients (on-off modules, adjustment of the clock). In general, techniques that reduce greater consumption have higher latencies. The saving strategies will have to take into account the time constraints: a real-time system with strict constraints will have to implement the saving strategies with shorter recovery times (less effective) than non real-time systems.

4.8 Bus Interface

A system bus is a device that connects several components of a electronic system. This technique was developed to reduce costs and improve modularity. The goal is based on management of data bus and address bus to determine what and where data must be sent; moreover, a control bus determine its operation.

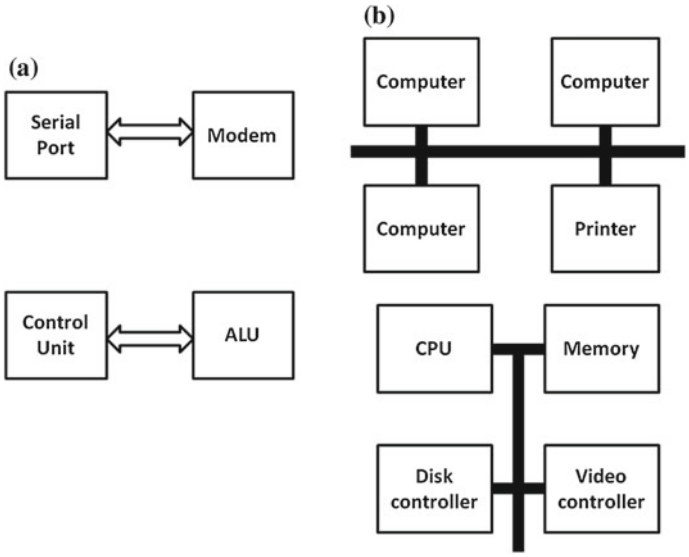


Fig. 4.13 Example of point-to-point buses (**a**) and multi-point buses (**b**)

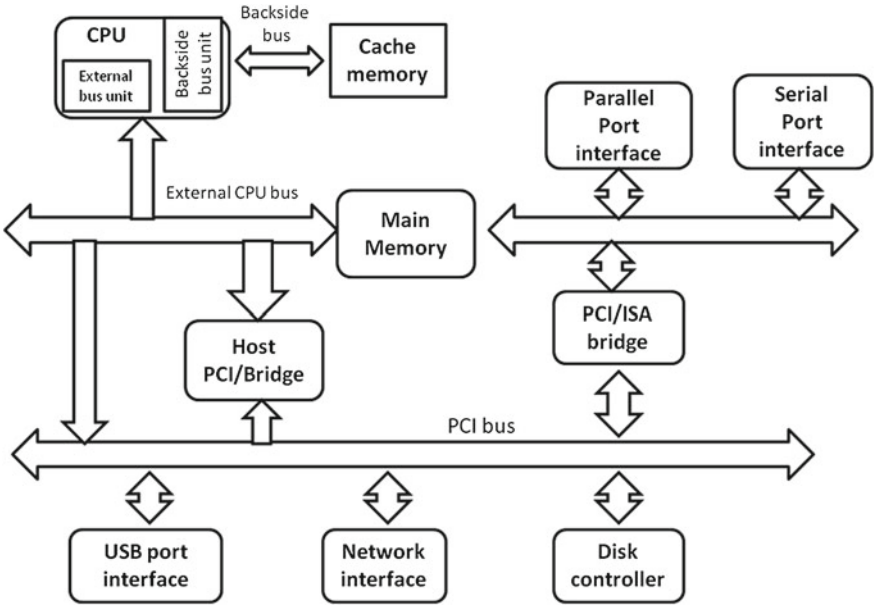


Fig. 4.14 Bus interface

In according to the Figs. 4.13 and 4.14 data are moved from several electronic devices by means of buses [2, 4–9].

Buses are designed in one of three ways:

a. Point to point connection: a particular bus is designed for every transfer.
b. Common bus: a common bus is used for all transfers.
c. Multiple bus: it is a combination of the previous two methods.

The goal of the bus is connect CPU and memory; it represents the main characteristics of the system.

Many CPUs use a set of pins for communicating with memory, but able to operate at different speeds with different protocols. Others used smart controllers to send the data directly in memory, this is known as Direct Memory Access. Moreover, most modern systems combine both solutions.

In modern systems high speed memory is built directly into the CPU, known as cache, using high-performance buses that operate at speed much greater than memory.

Such systems are architecturally more similar to multicomputers.

4.8.1 USB and FireWire

A USB system consists in a asymmetric design with host controller and multiple daisy-chained devices. It was designed to avoid of use plug expansion cards into computer's PCI bus. For many devices such as digital cameras, USB has become the standard connection. Today USB is the most electronic devices in use except for the monitors and high-quality digital video where require a higher data rate than USB as FireWire; even in the next years this limit will be solved.

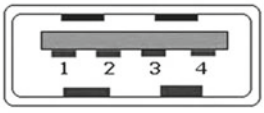
4.8.2 Standarization and Technical Details of USB Bus

The design of USB (Figs. 4.15 and 4.16) is standardized by the USB Implementers Forum (USB-IF), an industry standards body incorporating leading companies from the computer and electronics industries.

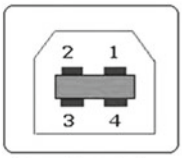
A physical USB device may consist of several logical sub-devices; its communication is based on pipes (logical channels) as visualized in Fig. 4.17. There are two types of pipes: stream and message pipes. Message pipes are used in bi-directional applications, for indicating status response. Stream pipes, instead, is used in unidirectional applications that transfers data using an isochronous, interrupt, or bulk transfer:

- control transfers;
- isochronous transfers;

pin	Name	Descryption
1	Vcc	+5 Vdc
2	D-	Data-
3	D+	Data+
4	GND	Ground



Type A socket
(from front)



Type B socket
(from front)

Fig. 4.15 Physical interface (USB)

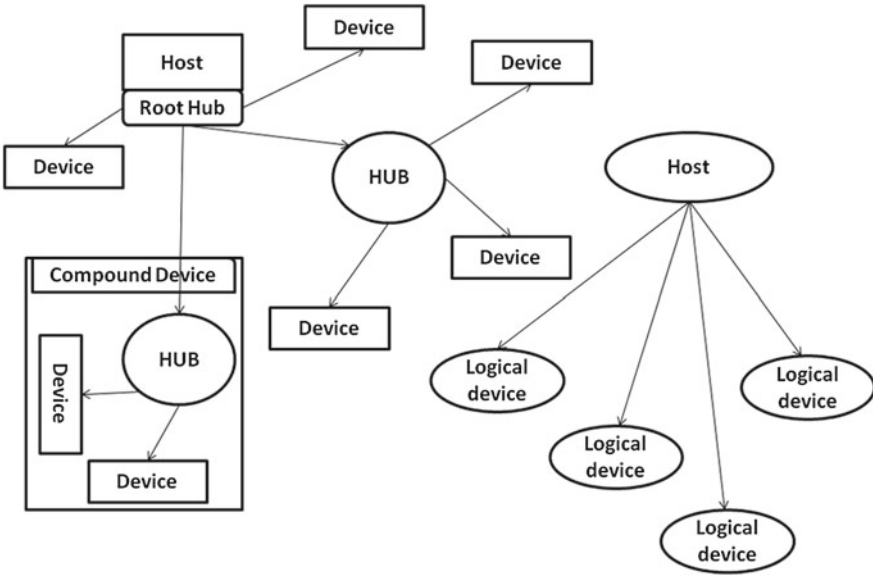


Fig. 4.16 Physical/logical bus topology (USB)

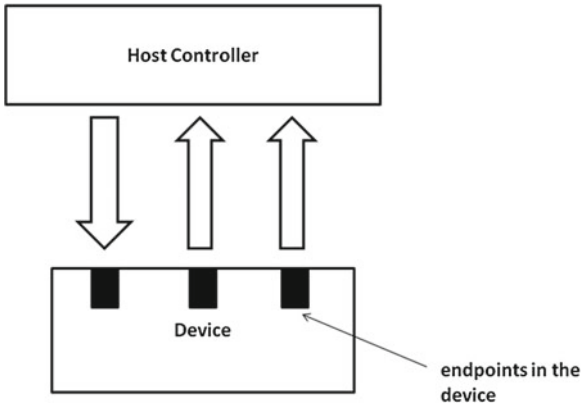


Fig. 4.17 Logical pipelines (USB)

- interrupt transfers;
- bulk transfers.

USB supports three data rates:

- 1.5 Mbit/s (used for human interface devices).
- 12 Mbit.
- 480 Mbit/s.

The connectors which the USB committee designed, are support for the following features:

- The connectors are designed to be robust.
- The connectors are particularly cheap to manufacture.
- The connectors enforce the directed topology of a USB network.
- The USB standard specifies relatively low tolerances for compliant USB connectors, intending to minimize incompatibilities in connectors produced by different vendors.

USB has become as much a standard for connecting power to portable devices as it has for serial communication. Recently the power aspects of USB have been extended to cover battery charging as well as AC adapters and other power sources. A tangible benefit of this wide-spread use is the emergence of interchangeable plugs and adapters for charging and powering portable devices. This, in turn, allows charging from a far wider variety of sources than in the past when each device required a unique adapter. Arguably the most useful benefit of USB's power capabilities is the ability to charge batteries in portable devices. Nonetheless, there is more to battery charging than picking a power source, USB or otherwise. This is particularly true for Li+ batteries, where improper charging can not only shorten battery life, but also become a safety hazard. A well-designed charger optimizes safety and the user experience. It also lowers cost by reducing customer returns and warrantee repairs [2, 4–9].

USB data is in the form of packets; all data are sent serially. Each USB data transfer consists of a:

- Token Packet
- Optional Data Packet
- Status Packet

Moreover, USB packets may consist of the following fields:

1. Sync field: All the packets start with this sync field.

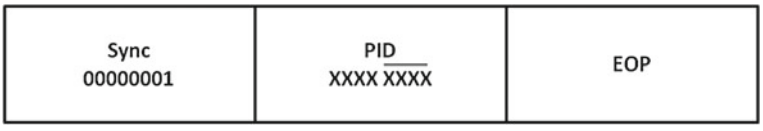


Fig. 4.18 Handshake packets (USB)

Fig. 4.19 Token packets (USB)

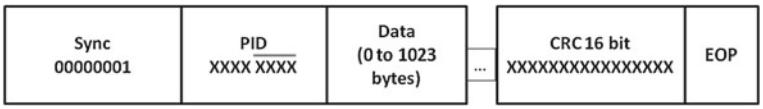


Fig. 4.20 Data packets (USB)

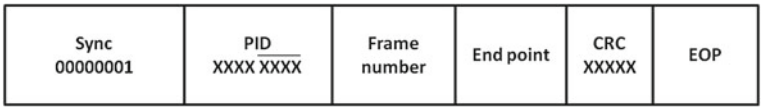


Fig. 4.21 Start of frame packets (USB)

2. PID field: identified the type of packet that is being sent.
3. ADDR field: specified which device the packet is designated for.
4. ENDP field: This field is made up of 4 bits, allowing 16 possible endpoints.
5. CRC field: Cyclic Redundancy Checks are performed on the data within the packet payload.
6. EOP field: indicates the End of packet.

The USB packets are defined in four basic types, each with a different format and CRC field:

1. Handshake packets, as visualized in Fig. 4.18.
2. Token packets, as visualized in Fig. 4.19.
3. Data packets, as visualized in Fig. 4.20. There are two basic data packets, DATA0 and DATA1.
4. PRE packet. Low-speed devices are supported with a special PID value, PRE.
5. Start of Frame Packets, as visualized in Fig. 4.21. Every 1ms, the USB host transmits a special SOF (start of frame). This is used to synchronize isochronous data flows.

FireWire was designed as high-speed serial bus for audio and video equipment. The main differences between FireWire and USB can be described in the following points:

- USB networks use a "tiered-star topology", while FireWire networks use a "repeater-based topology".
- USB uses a "speak-when-spoken-to" protocol, a FireWire device can communicate with any other node at any time.
- A USB network relies on a single host at the top of the tree to control the network. In a FireWire network, any capable node can control the network.

In other words, USB was designed for simplicity and low cost, while FireWire was designed for high performance.

4.8.3 Serial Communications

The RS-232 interface is the Electronic Industries Association (EIA) standard for the interchange of serial binary data between two devices. A typical RS-232 system (Figs. 4.22 and 4.23) is composed of two devices: data communicator (DCE) and Data terminal (DTE).

DTE stands for Data Terminal Equipment (for example a computer), and DCE stands for Data Communications Equipment (for example a modem). They are used to indicate the pin-out for the connection of the devices according to the direction of the signals.

In the following text we report the description of the signal in RS232 interface in according to the Fig. 4.23:

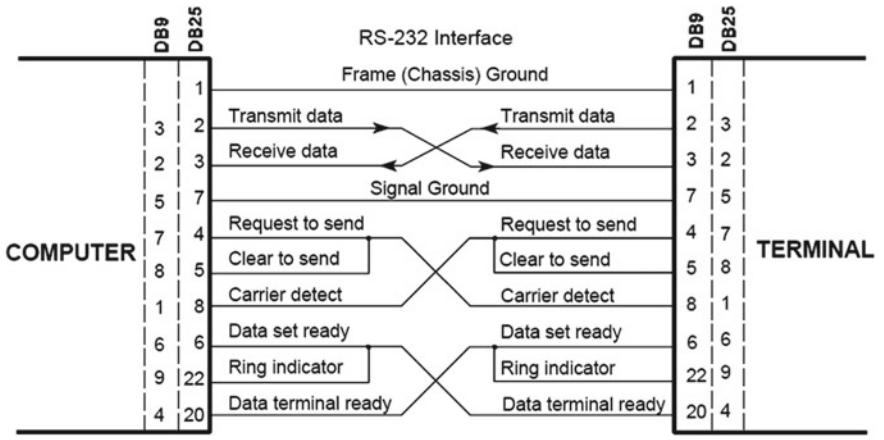


Fig. 4.22 RS232 interface

Pin number	Signal	Description
1	DCD	Data carrier detect
2	RxD	Receive data
3	TxD	Transmit data
4	DTR	Data terminal ready
5	GND	Signal ground
6	DSR	Data set ready
7	RTS	Ready to send
8	CTS	Clear to send
9	RI	Ring indicator

Fig. 4.23 RS232 interface - DB9

TxD: This pin carries data from the computer to the serial device
RXD: This pin carries data from the serial device to the computer
DTR signals: DTR is used by the computer to signal that it is ready to communicate with the serial device like modem.
DSR: Similarly to DTR, Data set ready (DSR) is an indication from the Dataset that it is ON.
DCD: Data Carrier Detect (DCD) indicates that carrier for the transmit data is ON.
RTS: This pin is used to request clearance to send data to a modem.
CTS: This pin is used by the serial device to acknowledge the computer's RTS Signal. In most situations, RTS and CTS are constantly on throughout the communication session.
Clock signals (TC, RC, and XTC): The clock signals are only used for synchronous communications.
CD: CD stands for Carrier Detect. Carrier Detect is used by a modem to signal that it has a made a connection with another modem, or has detected a carrier tone.
RI: RI stands for Ring Indicator.

RS-232 has some serious limitations as electrical interface. First of all, serial communications needs a common ground between DTE and DCE. This can be clearly in short cable, in an opposite way, with longer cables and connections between devices can be a problem causing "uncommon grounds". Moreover, as the baud rate and cable length increased, the effect of capacitance between cables introduces crosstalk noise which can be reduced by using low capacitance. In the end, RS 232 was designed for communication of local devices (30–60 m maximum), and supported one transmitter and one receiver [2, 4–9].

4.8.4 Wireless, Ethernet and Bluetooh

Ethernet communication (Fig. 4.24) consists of the division of a bytes stream into shorter pieces called frames. Each frames contains data source, address and error-checking data that controls when the data are damaged and enable the device to re-transmit data source. The frame structure (Fig. 4.25) consists of the following fields:

- Preamble: This allows the receiver's clock to be synchronized with the sender's.
- The Start Frame Delimiter: indicates the start of frame.
- The Destination Address: address where frame is sent.
- The source address: address of data source.
- Length: size of data in the ethernet frame.
- Data: the information sent by ethernet connection.
- Pad: to compensate the data transmission because frame must be at least 64 byte long.
- Checksum: used for error detection.

Wireless telecommunication known as IEEE 802.11 is a set of standard for implementing the transfer of information between two points that are not physically connected. The frequency bands are 2.4, 3.6 and 5 GHz. Other example of wireless technology are: GPS, bluetooth and so on.

The 802.11 family consists of a series of half-duplex over-the-air modulation techniques that use the same basic protocol. The most popular are those defined by the 802.11b and 802.11 g protocols, which are correct to the original standard.

Current 802.11 standards define "frame" types for using in transmission; each frame consists of a MAC header, payload and frame check sequence (FCS). The first two bytes of the MAC header form a frame control, it is further subdivided into the following sub-fields:

- Protocol Version: the protocol version.
- Type: the type of WLAN frame.
- Sub Type: addition discrimination between frames.
- ToDS and FromDS: They indicate whether a data frame is headed for a distribution system.
- More Fragments: The More Fragments bit is set when a packet is divided into multiple frames for transmission.
- Retry: Sometimes frames require retransmission, and for this there is a Retry bit which is set to one when a frame is resent.
- Power Management: This bit indicates the power management state of the sender after the completion of a frame exchange.

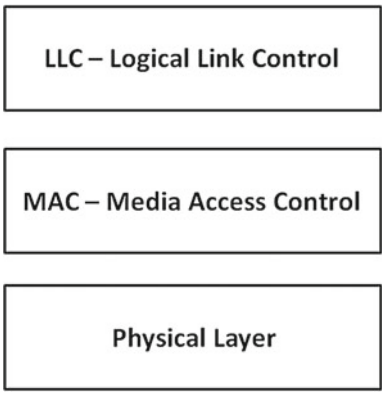


Fig. 4.24 Ethernet protocol

Preamble (8 bytes)	Destination Address (6 bytes)	Source Address (6 bytes)	Frame Length (2 bytes)	Data (0-1500 bytes)	Pad 0-46 bytes)	Checksum (4 bytes)

Fig. 4.25 Ethernet frame

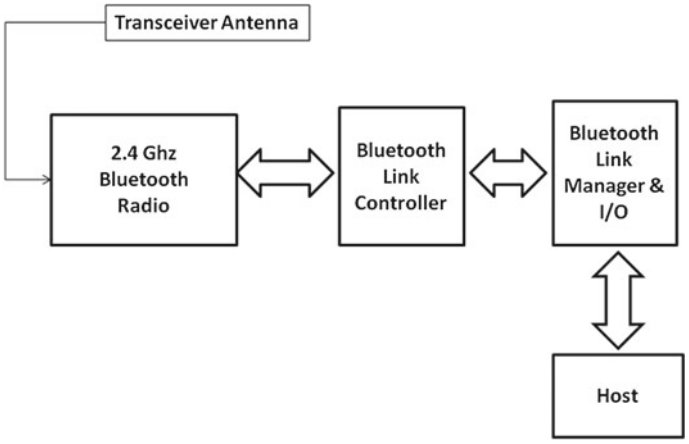


Fig. 4.26 Bluetooth

- More Data: The More Data bit is used to buffer frames received in a distributed system.
- WEP: The WEP bit is modified after processing a frame.
- Order: This bit is only set when the "strict ordering" delivery method is employed.

The next two bytes are reserved for the duration ID field. This field can take one of three following forms: Duration, Contention-Free Period (CFP), and Association ID (AID).

Bluetooth (Fig. 4.26) is a method for data communication that uses short-range. It is based on a nominal antenna power of 0 dbm. Each bluetooth stage has a free-running clock, CLKN, that determines timing and hopping of the transceivers.

4.8.5 GSM for Embedded System

GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile), is a communication standard developed by the European Telecommunications Standards Institute (ETSI) to describe protocols used by mobile phones. The network is composed of the following structures:

- The Base Station Subsystem (the base stations and their controllers).
- The Network and Switching Subsystem (the part of the network most similar to a fixed network).
- The GPRS Core Network (optional part).
- The Operations support system (OSS) for maintenance of the network.

GSM networks works in a different carrier frequency ranges, with most 2 G GSM network that operates in the 900 or 1,800 Mhz bands.

Most 3 G networks in Europe operate in the 2,100 MHz frequency band.

GSM uses a digital modulation format called 0.3 GMSK (Gaussian minimum shift keying). The 0.3 describes the bandwidth of the Gaussian filter with relation to the bit rate.

GMSK is a special type of digital FM modulation. 1's and 0's are represented by shifting the RF carrier by plus or minus 67.708 KHz. Modulation techniques which use two frequencies to represent one and zero are denoted FSK (frequency shift keying). In the case of GSM the data rate of 270.833 kbit/s is chosen to be exactly four times the RF frequency shift. This has the effect of minimizing the modulation spectrum and improving channel efficiency. FSK modulation where the bit rate is exactly four times the frequency shift is called MSK (minimum shift keying). In GSM, the modulation spectrum is further reduced by applying a gaussian pre-modulation filter. This slows down the rapid frequency transitions, which would otherwise spread energy into adjacent channels.

GSM uses TDMA (time division multiple access) and FDMA (frequency division multiple access). The frequencies are divided into two bands. The uplink is for mobile transmission and the down link is for base station transmission. Each band is divided into 200 KHz slots called ARFCN (Absolute radio frequency channel number).

Timing advance is required in GSM because it uses time division multiple access (TDMA). Since a radio signal can take a finite period of time to travel from the mobile to the base station, there must be some way to make sure the signal arrives at the base station at the correct time.

The GSM standards define a radio communications system that works properly only if each component part operates within precise limits. Essentially, mobiles and base stations must transmit enough power, with sufficient fidelity to maintain a call of acceptable quality, without transmitting excessive power into the frequency channels and time-slots allocated to others. Similarly, receivers must have adequate sensitivity and selectivity to acquire and demodulate a low level signal [4, 5].

4.8.6 PCI and PCI Express

The PCI architecture (Fig. 4.27) was designed as a replacement for the ISA standard, with three main features: increasing the performance during data transferring, to be independent, to simplify the system with the possibility of adding or removing peripherals. PCI architecture works with clock that run at 25 or 33 Mhz. Moreover, it is equipped with a 32-bit data bus and a expansion bus of 64-bit.

The PCI bus is the de-facto standard bus for current-generation personal computers and embedded applications. The main advantages for using in embedded applications can be the following:

- direct implementation in FPGAs
- efficient protocol
- ready availability of development hardware

In the Figs. 4.28 and 4.29 it is possible observe typical read and write signals transfer for PCI bus.

An improvement of PCI is called PCI Express (Peripheral Component Interconnect Express), officially abbreviated as PCIe. The main different between PCI and PCI Express are the following: more detailed error detection, smaller physical footprint, higher performance bus. In other words, PCIe can be seen as a high-speed serial replacement the older PCI bus. Another important difference is the bus topology: PCIe uses a shared parallel bus architecture [4, 5].

In terms of bus protocol, PCIe communication is encapsulated in packets.

Moreover, the stated goal of PCI-Express can be provided by:

- A local bus for chip-to-chip interconnects
- A method to upgrade PCI slot performance at lower costs

4.8.7 Compact PCI

The Compact PCI technology represents not only a very interesting bus application in the automation industry but also in the field of telecommunications. Mechanical components are used to work for standard connection applications to optimize high-performance systems. The industrial consortium PICMG (PCI Industrial Computer Manufacturers Group) has seen an architecture based on PCI but with similar mechanics, such as VME: the compact PCI (cPCI) is based on 3U and 6U Eurocard cards with pin connector 2 mm, which can serve up to 8 boards on a single bus (compared to 4 PCI).

Signal Name	Driven by	Description
CLK	Master	Bus Clock (normally 33MHz; DC okay)
FRAME#	Master	Indicates start of a bus cycle
AD[31:0]	Master/Target	Address/Data bus (multiplexed)
C/BE#[3:0]	Master	Bus command (address phase) Byte enables (data phases)
IRDY#	Master	Ready signal from master
TRDY#	Target	Ready signal from target
DEVSEL#	Target	Address recognized
RST#	Master	System Reset
PAR	Master/Target	Parity on AD, C/BE#
STOP#	Target	Request to stop transaction
IDSEL		Chip select during initialization transactions
PERR#	Receiver	Parity Error
SERR#	Any	Catestrophic system error

Fig. 4.27 Bus PCI—signals

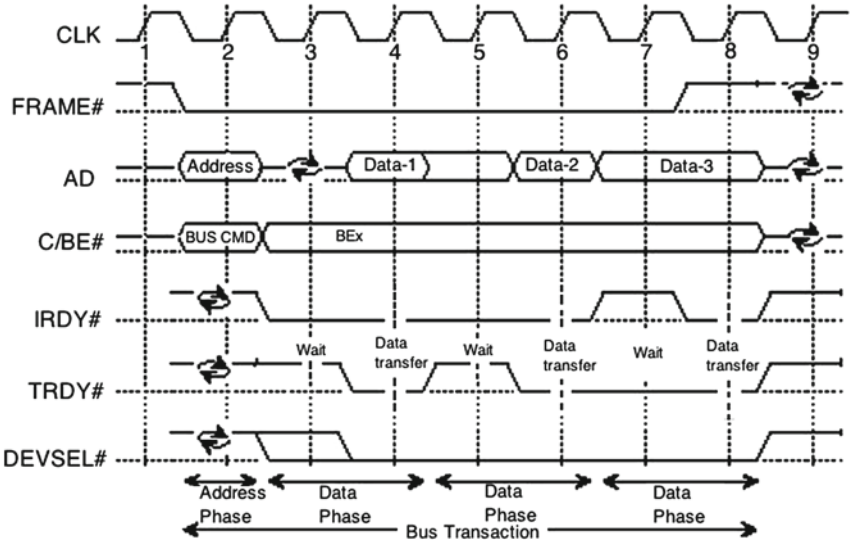


Fig. 4.28 Bus PCI—read

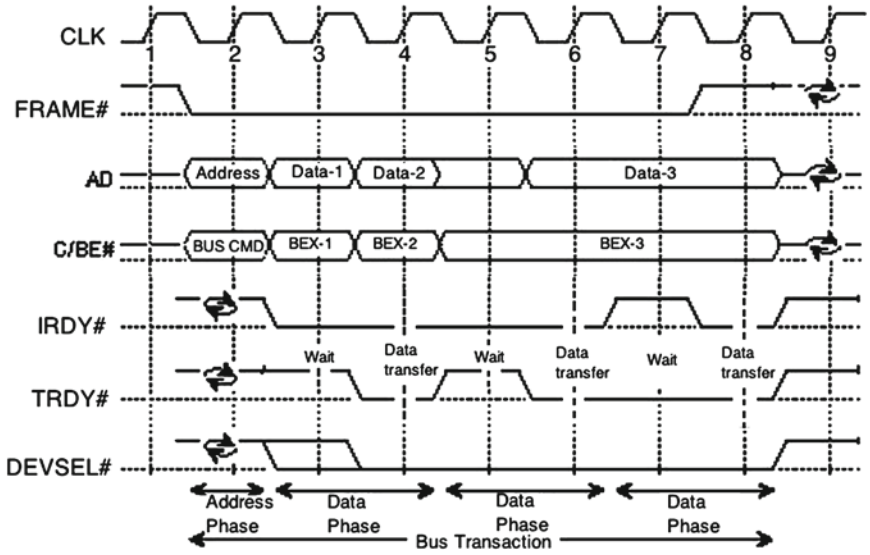


Fig. 4.29 Bus PCI—write

The Slot System provides arbitration, clock distribution and reset functions and is responsible for system initialization.

The Master System is equipped with a PCI-to-PCI bridge, while the Master Peripheral mounts a bridge placed directly on the CompactPCI card [4, 5].

The 2.x series of PICMG specifications provide to support for a wide range of technologies including hot-swap (PICMG 2.1) phone signals (PICMG 2.5), and in particular the expansion of the architecture to include Ethernet communication (PICMG 2.16).

The mechanical architecture has been defined at the beginning under the heading IEC 60297-3 (Fig. 4.30). The most recognized standards are IEEE 1101.1, IEEE 1101.10 and IEEE 1101.11. The IEEE 1101.10 is based on the requests of the EMI and is the necessary mechanics for SCREW 1.1-1997 (R2002) which is the standard for VME64 as PICMG 2.0 for the CompactPCI (Fig. 4.31).

The subrack have standardized dimensions both horizontally and vertically. The height is specified by the unit 'U', where 1U equals to 1.75 inches. The width is specified by the unit 'HP', where 1 HP is equal to 0.20 inches. In addition, the boards are produced with Eurocard modular, starting from 100 mm and increasing in steps of 60 mm.

The CompactPCI technology ensures high performance and independence of the PCI bus by the processor in a rugged Eurocard form factor and modular. The structure that is created is a solid technology for embedded computing and ideal for industrial controls.

The Emerson Network Power offers a well-structured series of cards for different applications: the CPCI7145 (Fig. 4.32) single-board computer (SBC) uses the Intel Pentium M and the E7501 chipset and offers high-speed server with advanced performance for telecommunications and control.

The single-slot SBC CPCI7145 is ideal for thermally constrained environments and includes two Gigabit Ethernet interfaces and dual channel 3.2 GB/s with a high-speed double data rate SDRAM.

Eurotech offers some embedded boards of communication in the format CompactPCI Express (cPCIe). They are characterized by the ability to have the port type of Gigabit Ethernet (10/100/1000 Base-T).

These cards are suitable for industrial applications, medical equipment, transportation and sercurity defense. In particular, the A3EXP1563 is a Gigabit Ethernet card that supports the CompactPCI Express standard and is equipped with two

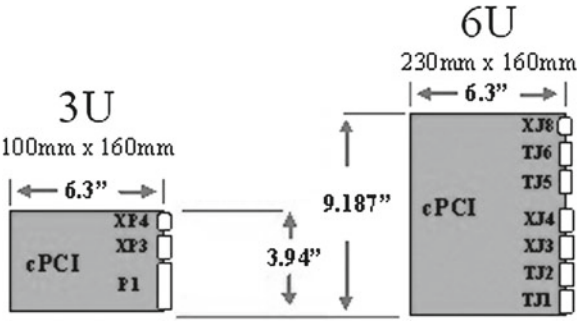


Fig. 4.30 Eurocard, cPCI

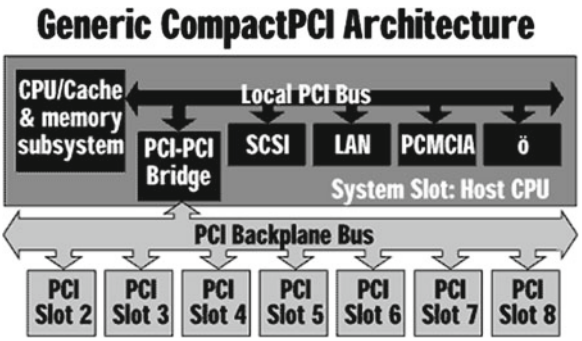


Fig. 4.31 Architecture of the CompactPCI

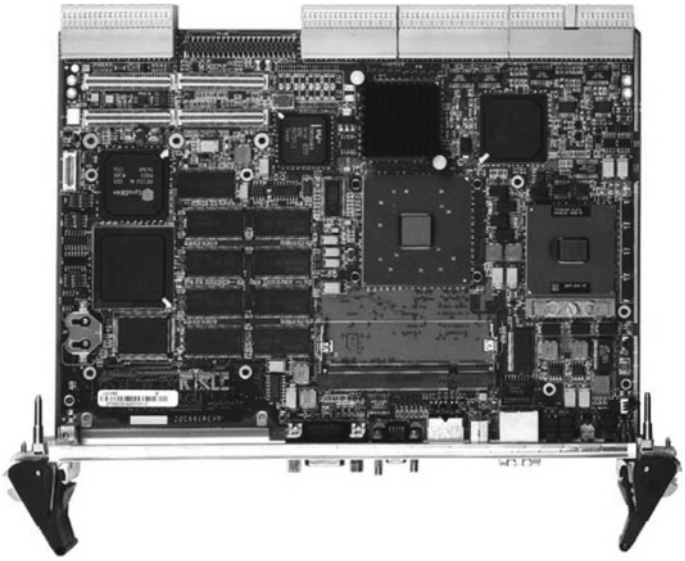


Fig. 4.32 CPCI7145

10/100/1000 Base-T Ethernet ports. The front panel is compatible with 6U and 3U form factor with 1 slot width.

National Instruments has put on the market time ago, the chassis PXI / Compact-PCI and PXI Express backplane PCB format.

Applications are in the field of defense and embedded applications. The robustness of the chassis provides excellent custom applications for every scientific requirement. Moreover, it is possible to use Labview software totally reliable. The packages of the chassis are available in version PXI / CompactPCI 3U and 6U with an expansion slot between 4 and 18.

4.8.7.1 Compact PCI Express

Compact PCI Express is a PICMG CompactPCI called EXP.0 released in 2005. The transition from CompactPCI to CompactPCI Express includes additional elements such as hot swap and system management. Features of CompactPCI, as the basic mechanics remain in the CompactPCI Express. The power supply is made with a 7—pin Universal Power Module, UPM , capable of delivering more than 400 W to individual modules.

4.8.7.2 PXI

PXI is a rugged PC-based platform, that applied in automation systems and combines the functionality of the PCI bus with the Europcard package.

PXI is the platform that has high performance and very low cost, ideal for military and aerospace applications. Developed in 1997 and subsequently launched in 1998, PXI has become an industry standard developed by the open PXISA (PXI Systems Allancé). The chassis is the main part of the PXI system and provides power, cooling, and communication between bus PCI and PCI Express. Each chassis has a wide range of configuration with different slot I/O and integrated peripherals.

4.8.8 Zigbee and RFID

The study of radio frequency devices is primarily concerned with the technology and its limitations, the potential, the fields of application, feasibility studies and any bureaucratic restrictions caused by specific areas of application [4, 5].

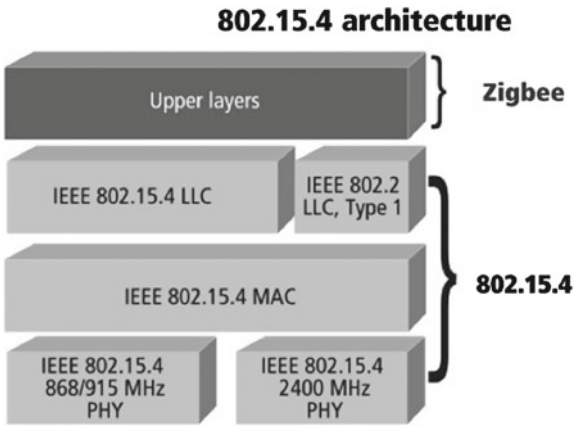
The ZigBee protocol has been designed for use in embedded applications that require a low data of transfer rates but especially low consumption. ZigBee defines a economic self-configuring wireless mesh network that can be applied in various fields including industrial control, home automation and telecommunications.

ZigBee (Figs. 4.33 and 4.34) is a wireless technology developed as an open global standard operating at the specification of the IEEE 802.15.4 radio frequency band of 2.4 GHz, 900 MHz and 868 MHz. Based on a packet-based radio protocol designed for battery-powered devices, the protocol allows devices to communicate in a variety of network topologies with long term batteries. The ZigBee protocol was created and ratified by member companies of the ZigBee Alliance. Over 300 leading semiconductor manufacturers, technology companies, OEMs and service companies are part of them.

The ZigBee protocol is designed to provide an easy way to use wireless data solution characterized by reliable and secure architectures. It was designed for the data communication through RF hostile environments that are common in commercial and industrial environments. Features ZigBee protocol can be the following:

- Support for multiple network topologies including point-to-point.

Fig. 4.33 Standard ZigBee



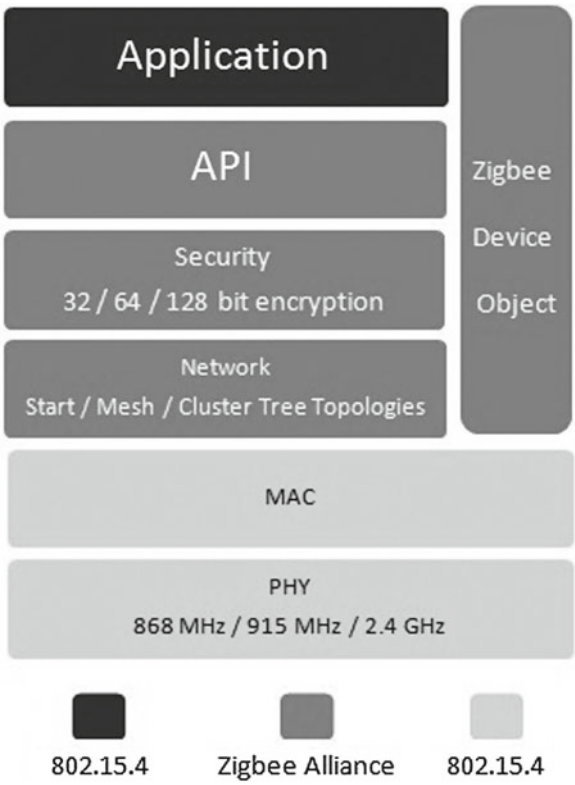


Fig. 4.34 Architecture of the ZigBee

- Mesh Network: type of network implemented with a wireless, decentralized, economic, adaptable and resistant.
- Low duty cycle.
- Low latency.
- Techonology of Direct Sequence Spread Spectrum (DSSS) for broadband transmission. Each bit is transmitted with a redundant sequence (chipping code). Suitable for weak signals.
- Up to 65,000 nodes per network.
- 128-bit AES encryption for secure data connections.

A key component of the ZigBee protocol is the ability to support mesh networking. In a mesh network, nodes are interconnected with other nodes so that multiple pathways can connect with each other. The connections between nodes are dynamically updated and optimized through sophisticated mesh routing tables. ZigBee is suitable for all purposes that do not require large data transfer speeds, up to 250 kbit/s in the applications of home automation and industrial monitoring where also require low power consumption.

The IEEE 802.15.4 standard (Fig. 4.35) allows the optional use of a superframe structure. The format of the superframe is contained between two beacon messages sent by the latter at regular and programmable intervals. These beacons contain information that can be used for the synchronization of the devices, for the identification of the network, to describe the structure of the superframe and to provide the periodicity of the beacons themselves.

The main elements that constitute an RFID (Fig. 4.36) identification system are the following: the tag, antenna miniaturized, reader and the software interface. Tags, depending on the scope, taking shape, size and various characteristics represented according to the type of power supply (active tags, passive or semi-active), the transmission frequency (UHF high frequency, mid-frequency MW, LF low frequency)

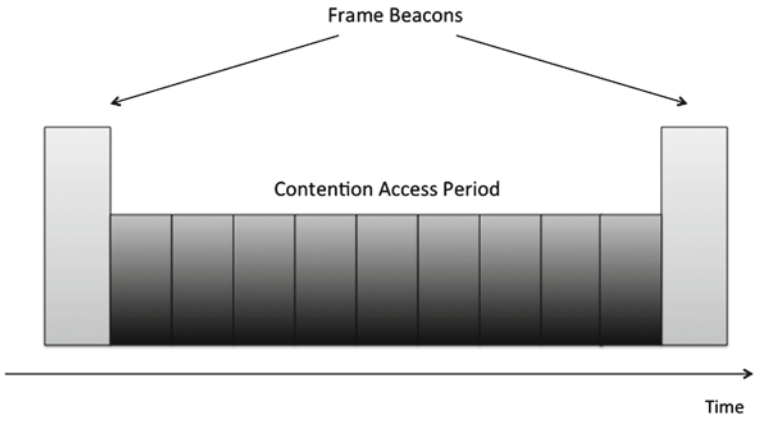


Fig. 4.35 ZigBee frame

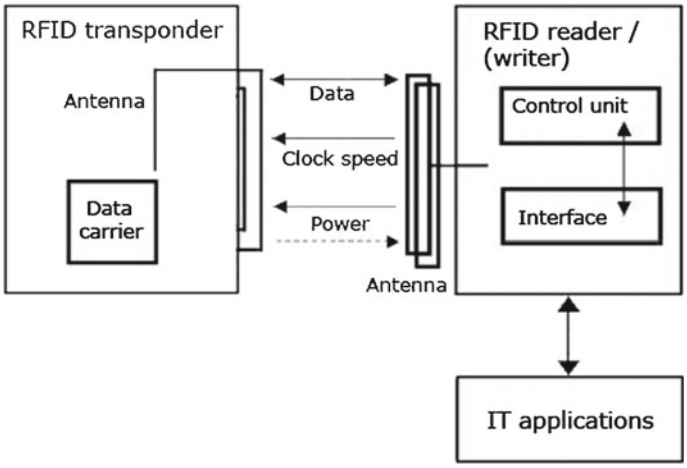


Fig. 4.36 RFID architecture

and the capacity and rewritability of memory, which can range from a few bits up to hundreds of kilobytes and can be either Read Only, Write Once & Read Only and Read & Write.

The Tag can work in two main modes: distributed architecture, which provides a range of information that are directly contained in the memory of the tag, or concentrated architecture in which the tag contains is only the identification code and therefore it is less economical. The antenna is the element responsible for providing energy to the tag and receive the radio signal.

The tag receives energy from the electromagnetic field produced by the antenna and transmitted from the player (in the case of passive tags); the antenna converts the electromagnetic energy in voltage and current that powers the chip of the tag; the chip then returns a modulated radio frequency signal that is received via the antennas themselves.

The readers are the links between the tag and the management system from which they receive instruction in reading and transmitting of read data.

The reader can be classified in according to the communication protocol, the working frequencies, the number of connectable antennas and transmission power.

The whole system of devices requires a special software interface that is used to connect the data on the tags to enterprise systems. The software is able to process, choose and combine the extracted data from tags through the reader and decide what information send to the server. Then, it connects the codes to other available information by querying a server that uses a language PML (Physical Markup Language) based on the XML standard [4, 5].

4.9 Memory

Modern microprocessors can manage a huge amount of data in short time.

Memory is a large array of bytes, each with its own address. CPU fetches instructions may cause loading from and storing to specific memory address. A instruction-execution cycle can be composed of:

- Fetches an instruction from memory.
- Decodes an instruction.
- Fetches operands from memory.
- Executes the instruction on the operands.
- Stores results back in memory.

Basic operation cycle of a computer is defined as instruction cycle, sometimes called fetch-and-execute cycle. This cycle is repeated continuously by the central processing unit (CPU), from bootup to when the computer is shut down.

Memory management (Fig. 4.37) is the process of computer memory. The main requirement is to provide ways to dynamically allocate area of memory to programs in request, and freeing it when is demanded. This operation is critical to the computer system. Several methods have been devised that increase the effectiveness of memory management. Virtual memory is a method of decoupling the memory from physical hardware. It permits the separation of processes by increasing the efficiency of the memory. Its goal is to order the translated virtual address to a physical address.

In this way, the additional virtual memory enables control over memory systems and methods of access:

- Protection: it is used to disable the process to read and write to memory that is not allocated.
- Sharing: it is used for Inter-process communication.

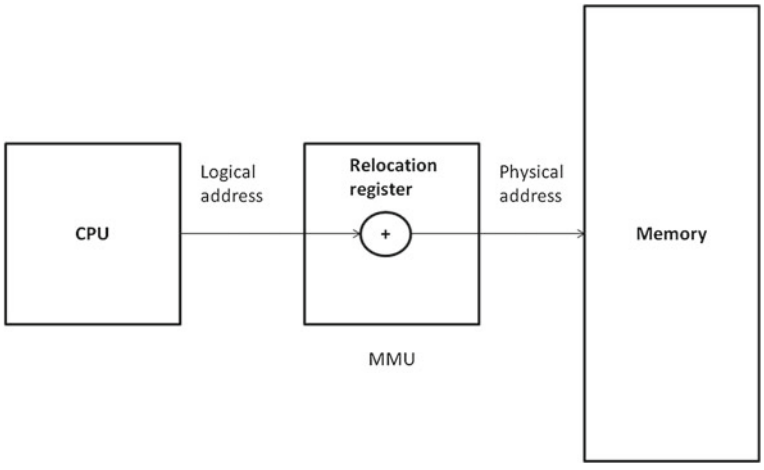


Fig. 4.37 Memory management

4.9.1 Memory Flash

Flash memory was developed for the first time from Intel (1988), it is a semiconductor memory built in CMOS technology as part of the class of non-volatile memories and reprogrammable (NVRWM , Non-Volatile Read Write Memory).

Togheter EPROM and EEPROM, flash memory are parts of the NVRWM class; all three memories (EPROM , EEPROM, FLASH) are composed of the cell floating gate transistor FAMOS (Floating-gate Avalanche-injection MOS (Fig. 4.38), with the same mechanism for programming.

The Flash memory is a type of low cost portable memory that does not need of power to maintain the data, and occupy little space but with a good capacity of memory.

For all these good properties, the application are such as digital cameras, cell phones, in the pendrive, in microprocessors and many other devices.

The flash memory cell is based on a double-gate MOSFET device (FAMOS), i.e. a conventional MOS transistor with an additional layer of polysilicon added between the gate and channel, is called Floating-Gate, which is isolated by the silicon oxide and capacitively coupled with the electrode relative to the control gate (Gate-control), as is shown in Fig. 4.39 [4, 5].

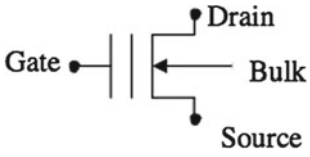


Fig. 4.38 FAMOS, electrical symbol

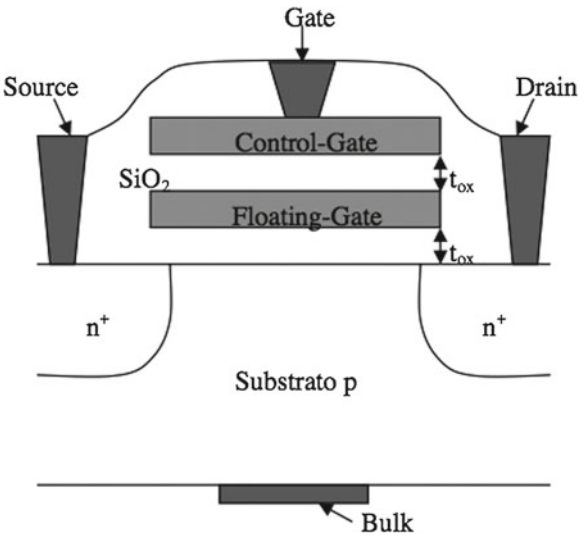


Fig. 4.39 Floating-gate (FAMOS) transistor

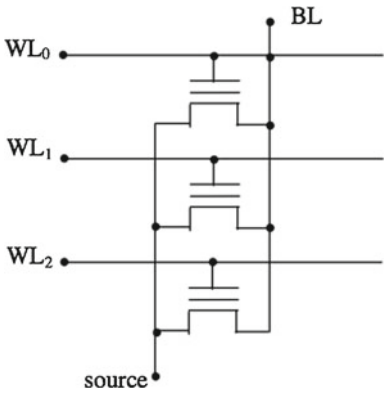


Fig. 4.40 Memory flash: NOR architecture

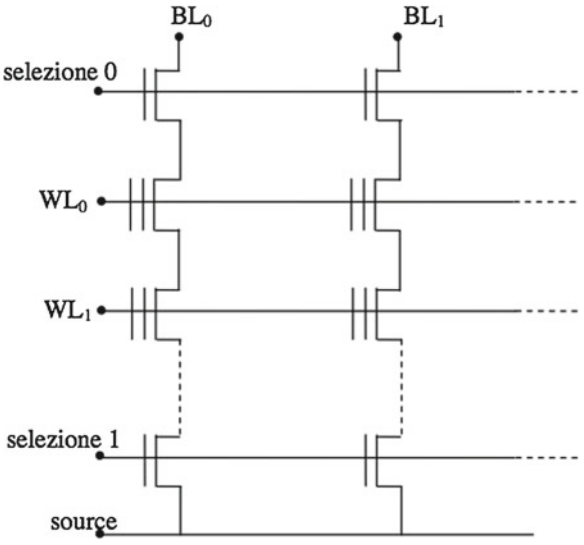


Fig. 4.41 Memory flash: NAND architecture

Today, the flash memory architecture with NOR gates are those most mainly in used and in the fields that require low density of data, for example, the operating systems of digital cameras or mobile phones.

The NOR architecture is a very versatile structure that allows to support different modes on the same chip.

As shown in Fig. 4.40, the Control-Gate of the cells in the same row which is connected by the same word–line; the drain of the cells on the same column is connected by the same bit line and the source of the cells in the same sector which is connected by a common line—source.

Every single cell is addressable through two signals, one is coming from the bit-lines and another is from the word lines. The cell is addressed via the row, while

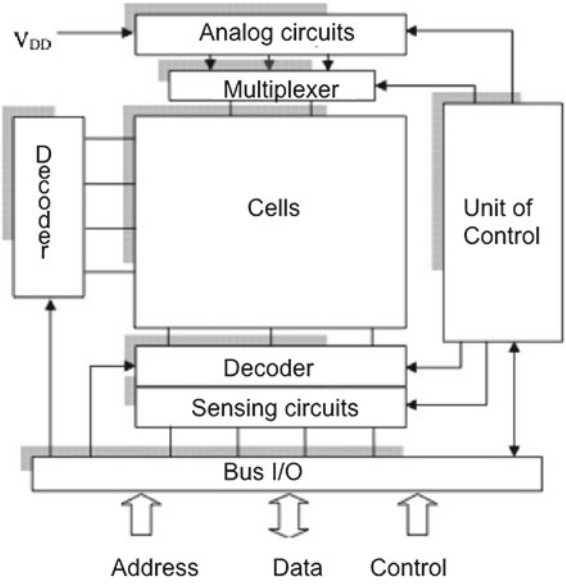


Fig. 4.42 Memory flash: general architecture

the Source-Switch is used to apply the voltage during erasure. Finally, each column is connected to the output and the amplifier which recognizes the information content of the cells.

The access times are less than a NAND architecture (Fig. 4.41), because the operations are not sequential.

Flash memory is formed over the array of cells that also serves as a peripheral circuitry to carry out all the features of the device, as shown in Fig. 4.42.

References

1. Park, J., & Mackay, S. (2003). *Pratical data acquisition for instrumentation and system control*. Boston: Elsevier.
2. Razavi, B. (2008). *Fundamentals of microelectronics*. New York: Wiley.
3. Ganssle, J. (2008). *The art of designing embedded systems*. Oxford: Newnes.
4. Kang, S. M., & Leblebici, Y. (1998). *CMOS digital integrated circuits*. New York: McGrawHill.
5. Di Paolo Emilio, M. (2013). *Data acquisition system from fundamentals to applied design*. New York: Springer.
6. Valdes-Perez, F.E. & Pallas-Areny, R. (2009). Microncontrollers, fundamentals and applications with PIC. Boca Raton: CRC Press.
7. Harper, A. (2000). *High performance printed circuit boards*. New York: McGrawHill.
8. Thierauf, S. C. (2004). *High-speed circuit board signal integrity*. Boston: Artech House.
9. Ellis, G. (2004). *Control system design guide*. Oxford: Elsevier.

Chapter 5
Embedded Development System and C Programming

Abstract An embedded system is identified as the electronic processing device designed for a certain function. The designers usually use compilers, assembler, debugger, and a whole range suites for the development of both software and hardware in it.

5.1 Development System

Embedded system designers usually use compilers, assembler and debugger, but also can refer to the design with even more specific programs such as:

- An in-circuit emulator (ICE): it is an interface of microprocessor for quickly debugging the code.
- MathCad or Mathematics: to simulate the mathematics in the context of systems by use of DSP (Digital Signal Processor).
- Compilers and Linkers that optimize the hardware.

An embedded system may have its own specific language, development program or offer improvements to an existing language. The increasing of the system complexity requires hardware support (or BSP Board Support Package) for easy integration with the software and the operating environment. The programs for the generation of the software may have different origin, in particular:

- Tool Open Source.
- Tool for PC.
- Production Companies.

In the field of software, a tool chain is the set of programs (tools) used in the product development. The tools can be used in the chain, so that the output of each tool represents the input for the next. The term is still used more widely to refer, in general, set of development tools linked altogether. The development tools are available in GNU/Linux environment which consist of a native tool chain performed on the x86 workstation with generation code. For the development of embedded systems, the accessibility to use a native tool chain is impossible; in many cases, however, are less interesting for the following reasons:

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_5

- Usually, the target has a limited quantity and at times somehow is for its limited storage and/or memory.
- The target is very slow compared to the workstation.
- Unwanted to install all the development tools on the target board.

Therefore, it is used the cross-compiling tool chain performed on the workstation, which allows generating the code for the target. There are many different microcontrollers with free tool chain, these are included:

- Atmel AVRMega and AVRTiny (the heart of the Arduino platform).
- Atmel AVR32 (which includes a Linux kernel).
- Texas Instruments MSP430.
- Axis 100LX, used in Foxboard.
- ARM7 and beyond. ARM9 is included is a Linux kernel.

Examples of hardware where these microcontrollers can be the following: Raspberry PI, a single-board computer developed in the UK by the Raspberry Pi Foundation towards the end of February 2012; Arduino, an open source framework that allows quick design and rapid learning of the basic principles of electronics and programming; FOX Board G20, a micro Linux system made in Italy designed by Acme Systems in the form of small size with Linux pre-installed and running [1, 2].

5.2 C Programming

The C programming language was originally developed by Dennis Ritchie of Unix systems. It is often called a programming language for medium/high level as combines the elements of high-level languages with the functionalism of assembly language. One of the best features of C is that it is not tied to any particular hardware or system. This makes it easy for a user to write programs that will run without modification on almost all the virtually machines. Most common programming languages for embedded systems are C, BASIC and assembly languages. C used for embedded systems is slightly different compared to C used for general purpose (under a PC platform). Programs for embedded systems are usually expected to monitor and control external devices and use the internal architecture of the processor such as interrupt handling, timers, serial communications and other available features. There are many factors to take into consideration when selecting languages for embedded systems:

- Efficiency, programs must be as short as possible and memory must be used efficiently;
- Speed, programs must run as fast as possible;
- Ease of implementation, maintainability and readability.

C compilers for embedded systems must provide ways to examine and utilize various features of the microcontroller's internal and external architecture: for

example, interrupt Service Routines, reading from and writing to internal and external memories, bit manipulation, implementation of timers/counters, examination of internal registers. Most embedded C compilers (as well as ordinary C compilers) have been developed to support the ANSI [American National Standard for information] but compared to ordinary C, they may differ in terms of the outcome of some statements. Obviously that the trend is doing to migrate from developers of embedded systems in assembly to C language. Unlike commercial systems, embedded systems don't leave space for an error of choice.

The choice to use the C language for applications based on embedded computing platforms, is a choice that can lead to risks of failure. Embedded systems in the mostly cases are based on programmable devices such as microcontrollers, the digital signal processor, graphics processors and so on. Until a few years ago, the most efficient programming of these processors was only one made with assembly language, the only one with a direct correspondence between symbolic instructions and machine instructions. In this way, assembly language ensures that the code is minimal, both with respect to the occupation of memory and execution time. Minimum occupancy of code and minimum execution time are typical of the embedded systems requirements that the developer has to respect. Usually, it is therefore forced to develop the application in assembly language, especially relating to the critical parts such as device drivers; functions that directly govern the hardware of the system; the procedures for managing interruptions and so on. In fact, the developer does not easily trust the high-level language, as it fails to have the complete knowledge of the correspondence of what it writes on a symbolic level and how the compiler actually produces at the level of machine instructions.

5.2.1 From Assembly Language to C Language

The C language in the development of commercial systems has been used since the introduction of the first personal computer. For decades, programmers of embedded systems have ignored the C language for compiler problems, despite of its spread. From time to time, some producers of C compilers released a version for microcontrollers or DSP, but the quality of the code produced was too low to induce developers to discard the possibility of using the C language in the development of embedded systems. One of the main problems that the C compiler must be able to handle for embedded systems is to overcome the limited precision of integer (arithmetic) and of the ALU. The integer value, such as typical 8 and 16 bit microcontrollers, requires a lot of attention of the real numbers and their arithmetic manipulation is implemented on such architectures. it's very abstract for C compilers to computing the architecture without a compile-optimizing model. Some C compilers make for this purpose, extensions that allows to specifically supporting fixed-point arithmetic embedded processors and in the use of variables such as hardware register. However, there is a strong motivation to shift from assembly language to C language for embedded systems. The main motivations are the portability and readability. Moreover, rapid application development with C warrants more easily the requirements

of time-to-market. These issues become further significant if the code needs to be transferred from one developer to another, and must be maintained for a long time. Obviously, it can only be met if the executable code is efficient and compact, which can only be guaranteed by an optimized compiler.

5.2.2 Choose the Right C Compiler

Choose the right C compiler for the development of embedded applications is definitely the most important step when decides to move from assembly programming in the C embedded compilers, it's also available from C to C++ compilers. Quality Compilers are available even by non-commercial organizations as shareware. The best-known case of the GCC GNU compiler is shareware, and often used by manufacturers of microcontrollers as reference of compiler for their families of devices (comes in the form of free development tools). The criteria to keep in mind for choosing the C compiler are as the following:

- ANSI compatibility;
- Portability;
- Ability to generate compact code.

The compatibility of Ansi is closely related to the portability of the code. This is a direct consequence of the nature high-level C code and theoretically, any C compiler can compile any code C; in reality, this is not true for portability problems on source code. The ability for C compiler to optimize is the main feature that allows executing compact and efficient code.

The compiler should have at least the ability of optimization: a good optimizing compiler is also able to offer an action of weight optimization. The portability of the code is unique for final products that performs the application functions and subject to rapid and frequent upgradability. The development environment is another important aspect, do not forget to consider the complexity of embedded applications and hence the need for productivity tools during the design phase, but also during the verification, validation and documentation of the code. In the basic version, it is normally included in the EVM as a free basic version. The richest and most complete versions are in charge. A commercial development environment that includes the C language in a development suite for embedded systems is Metrowerks CodeWarrior. This development environment, especially since it was acquired by Motorola (later Freescale), it is particularly oriented to support the embedded devices such as the wide range of microcontrollers and DSPs.

5.2.3 ANSI C or C++

The C language has evolved from the original version, Ansi C, to other versions which are able to meet the growing and various needs of designers: graphical interfaces, communication, interaction with external devices and so on. New versions of

C were developed to meet the growing demand of programmers, particularly, are those using programmable embedded systems such as the microcontroller. The C++ language, namely Objective C, has been one of the most important innovations of the C language (ANSI). The developers of embedded systems, which are using advanced computing platforms such as the ASP (Application specific processor), there were doubts as to whether the C++ language would be beneficial for the development of embedded applications and also with stringent requirements. Doubt greater than C++ is concerned a potential overload that this language produces its own mechanisms for implement the objects. The C++ language should be considered as an extension of C with objective oriented mode. A commercial development environment that includes the C language in a development suite for embedded systems is CodeWarrior. CodeWarrior is an integrated development environment (IDE) for creating software on a number of embedded systems. The CodeWarrior IDE, is a suite that provides tools for developing complex software. The standard available tools in the IDE are: a project manager, editor, search engine, a browser, a build system and a debugger. The most important part of Code Warrior is the section of the simulator, which together with the debugger allows to test the newly created program without reference to any hardware [3, 4].

5.3 Code Warrior IDE

Demo versions are divided into two categories: Evaluation Edition, fully functional but with 30 days validity (except for insertion license code) and the Special Edition with the only restriction in the maximum size of the code can be generated. The Evaluation version is available for both Windows and Linux and has a high range of Freescale microcontrollers libraries. The Code Warrior IDE has several advantages instrumentals including the following points:

- Cross-Platform Development: develop the software to run on multiple operating systems or use the multiple hosts to develop the same project. The IDE works on the most popular operating systems, including Windows, Macintosh, Linux and Solaris, using virtually the same graphical user interface (GUI) in all hosts.
- Support multi-language programming: it is possible to choose from the most important programming languages to develop the software. The IDE supports high-level languages such as C, C++ and Java, as well as assemblers in line for most of the processors.
- Development environment: import of new processors. The IDE supports many families of embedded processors, including x86, PowerPC, and many others.
- Support of plug-in Tools to extend the functionality of the IDE: the IDE currently supports plugs-in for compilers, linkers, pre-linkers, post-linkers, preference panels. The plugs-in enables the CodeWarrior IDE to develop different programming languages.

The standard tools available in the IDE are as follows: a project manager, editor, search engine, a browser, a build system and a debugger. Particularly, the Web browser allows to manage a database with assistance in navigating the code, connections of each symbol and other locations in the relevant code. For each level of design, the management includes the following key points:

- Order of connection,
- Addictions,
- Compiler, linker and other settings.

The project manager (Fig. 5.1) automatically updates the target of code generation, also coordinating the program with information from the build target to call to the appropriate tools in the correct order with the specified settings. For example, the project manager directs the build system only for the source files that are based on information in a modified file. Note that all is done automatically. The software developer does not need to remember the syntax or semantics of the makefile and should never debug syntax errors. The IDE simplifies the process and makes it easier for software development. The project manager also supports multiple objectives of compilation within the same project file. Each build target can have its own unique settings and even use different source and library files. For example, it is common to have the debug and release build targets in a project.

The project window allows organizing the files of various aspects lists all the files for all targets. The window includes the following main items: the toolbar, tabs and columns. The Files page displays information about the files in a project. The items that can be included are the following: Text files, any files that contain text; Source file, file containing the source code and Library files, files that contain special codes to work together with a programming language or development environment.

The most important part of Code Warrior is the section of the simulator, which together with the debugger allows testing the program. The Visualization Tool offers a wide range of choice of software objectives for giving to the user the necessary control on an input or output of the microcontroller by a real point of view. The program can be tested without necessity to transfer directly into the program memory of the microcontroller. In this way, most of those software bugs can be eliminated. The hardware bugs are another problem in the design phase, the designer will have to be taken into account how to avoid them [4–7].

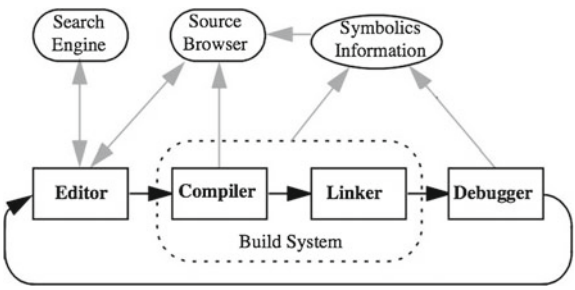


Fig. 5.1 Code warrior

5.4 Commercial Software

5.4.1 Labview Embedded

LabVIEW offers a full suite of design, prototyping of embedded systems, includes various pre-compiled libraries with good hardware integration and different approaches in the development graph, script files, connectivity C and HDL code. LabVIEW allows using the standard libraries with Xilinx CORE Generator IP Integration Node. In addition, NI LabVIEW C Generator allows creating ANSI C code and integrating with third-party tool chain with the possibility to download on any kind of target.

5.4.2 Intel Studio

Intel System Studio is a complete, integrated suite that provides developers with advanced tools and technologies for Embedded and Mobile system, helping to increase efficiency and high performance. Specially, Intel System Studio 2014 allows developing for Android Mobile and Embedded Systems with added of extended support JTAG for debugging for all platforms IA.

5.4.3 Altera

The Embedded design suite (EDS) provides tools for SoC Altera Embedded software development on SoC devices. It also includes utilities, software execution and examples to speed up the firmware.

5.4.4 IAR Embedded Workbench

IAR Embedded Workbench is the integrated development environment with a cross-compiler C/C++ for 8, 16 and 32 bit architecture with the IAR C-SPY debugger. IAR Embedded Workbench supports the following architectures: ARM, Atmel AVR, AVR32 Atmel, Freescale ColdFire, Freescale HCS12, S08 Freescale, Maxim MaxQ, National CR16, 78K Renesas, Renesas H8S, M16C/R8C Renesas, Renesas M32, R32 Renesas, Renesas RX Renesas RL78, RH850 Renesas, Renesas V850, Renesas SuperH SAM8 Samsung, STMicroelectronics STM8, TI MSP430, 8051. IAR C-SPY debugger is a language for high-level simulator and hardware support for the main emulators available on the market.

References

1. Park, J., & Mackqy, S. (2003). *Pratical data acquisition for instrumentation and system control*. USA: Elsevier.
2. Razavi, B. (2008). *Fundamentals of microelectronics*. Brazil: Wiley.
3. Kang, S. M., & Leblebici, Y. (1998). *CMOS digital integrated circuits*. New York: McGrawHill.
4. Di Paolo Emilio, M. (2013). *Data acquisition system from fundamentals to applied design*. Berlin: Springer.
5. Valdes-Perez, F. E., & Pallas-Areny, R. (2009). *Microncontrollers, fundamentals and applications with PIC*. USA: CRC Press.
6. Schildt, F. H., & Pallas-Areny, R. (2008). *C++ programming cookbook*. New York: McGrawHill.
7. Frescale, (2003). *Code warrior user guide*. Canada: Metrowerks.

Chapter 6
Real Time Operating System (RTOS)

Abstract A real-time system relies on the correctness of the processes execution time; system error is considered the violation of timing constraints. A real-time operating system (RTOS) is a crucial element on the Embedded system, but not the only one. It must ensure that each critical process takes place within the limits, using algorithms tailored to the demands of the system.

6.1 Operating System

A program is a sequence of elementary instructions that can be executed by the processing system; each program works on a set of information that constitutes the input and provides results. A program in execution is called process.

It is defined as the set of software programs that can be operated with the computer, i.e. the logical component of a computer, as opposed to the physical part in the hardware. The software of a computer system is normally divided into two categories:

- Basic software: dedicated to the management of the basic functions of the computer; this software works directly on the physical layer (hardware) machine;
- Application Software: dedicated to the creation of the specified applications.

The main component of basic software is the operating system, the program deputes to manage the various physical resources on the computer performing different tasks according to the complexity of the system under its control.

The operating system (OS), pratically, can operate on two levels:

- Management of the resources of the computer system:
  - Processor,
  - Main memory,

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_6

  - Mass storage,
  - Input/output.

- Provides the main interface on the machine.

The operating system of a computer and then the set of programs that allow the elementary operations of the machine (for example, reading and writing from and/into memory), and support the application programs and check for special events errors.

The operating systems can be divided into mono-tasking and multitasking in dependence of the fact that the processor can execute only one job at a time, or that it is able to handle more tasks simultaneously. The real-time multitasking operating systems are able to optimize processor utilization while giving the user possibility to allocate all resources for their work in the systems.

The result executed by a small portion of all processes are waiting to for next process; also if a job requires a long operation of the peripheral device, the CPU is allocated to the next command and the control returns to the process when the operation of input/output is suspended. This technique is called time-sharing because the CPU time has been divided between the various processes that contribute to its use; time interval (time slice) is so small that they can not be perceived by the user. There is also another technique called real-time multi-tasking processing, which alternates between running processes by the priority. This mode is used when some operations are so important that they can not be interrupted. The basic components of an operating system are as the following:

- Process management: a process is an activity's internal system related to the implementation of a program. At launch, the process creates in the memory structure which is necessary for the execution of the program (stack, data area , etc…).
- Memory Management: the system must take care to select the space to allocate what and when is required by the applications.
- Peripheral I/O: this is the most complex part, because the system must be able to handle a large number of external devices.
- File management.
- Protections.
- Management of interrupt.

The operating system controls all of the machine's resources, and only the software is able to do so. For this reason, all activities (by the user or applications) that involve these resources, must necessarily pass through the OS. This is possible to do in two ways:

- Interactive mode, using the operations provided by the shell (create files, run program, etc…).
- Programmed mode, using system calls, called syscall.

6.1.1 Classification of Operating Systems

6.1.1.1 Dedicated Systems

The dedicated system are single-user operating systems mono-programmed (used by one user at a time).

6.1.1.2 Batch Systems

The batch systems operate in the following ways: enter the system into all the input data, processing and print output. Theoretically, it is possible to batch systems interactivity with the user, but it is limited with the use of resources. In fact, sending all the data at the beginning, the CPU does not have to wait for more data and can performs its own processing with reduced downtime.

Users can prepare the data in offline mode (devices unconnected) and then the other peripheral equipments will upload the data online later. The step sequence to be performed is called job stream. This is sent from the computer to device mass channels in a queue controlled by the operating system. After processing, the output will send to a print queue associated with a processor that corresponds to the output channel selected by the user.

6.1.1.3 Multiuser Interactive Systems

The interactive systems aim to provide each user a device like the video, thus having the impression of a dedicated system. For this purpose, we use the time-sharing of the CPU: the CPU is allocated by each user for a set time (time slice). To avoid overloading, it's necessary to perform each time slice a number of CPU operations, which must be greater than the number of assignment instructions. The time slice is shorter means that the more time is dedicated to the routines of the operating system for the CPU reassignment, which can cause the overhead in the system, i.e. the CPU is too busy. In these cases, it's important to measure the CPU's speed in MIPS (millions of instructions per second), check the size of main memory where they are loaded into the operating system, users operating programs and data.

6.1.1.4 Transactional Systems

These systems are used to manage transactions and update the database. A transaction consists of several operations performed all to the end; otherwise, a rollback will be required (cancel all).

6.1.1.5 Real Time Systems

A real-time system requires the fast responding to stimulate the signals externally. Unfortunately, it will never respond instantly because the process requires the authorization before executing. These systems are useful, for example, to report an error when an abnormality occured in the aircraft engine or in the nuclear power plants, where the fast response of the systems is absolutely demanded [1–3].

6.2 Real Time Software

The nature of the system-on-chip programmable microelectronic devices has shifted the focus of developers from hardware to software, so that the software development of an embedded system has become predominant from 70–80 %, up to 90 % of the development effort. In the recent years, the complexity of embedded systems and the application has grown enormously. Furthermore, most of the embedded systems are real-time nature, with the stringent reliability requirements, for example, in the traditional embedding application such as automotive, communication, medical and consumer industry. In the past, the developers have addressed the problems of the system and development of real-time custom code to ensure the performance and reliability, but its now a common request in the application because of the promptly completion of the development (time-to-market). The real-time operating systems (RTOS) are therefore the natural solution to this problem, which can be basically a proprietary or open nature (open source). Processor manufacturers have been seeking to promote those of open nature, so as it does not have any impact on the development costs. In addition to those promoted by the processors manufacturers, there are numerous available solutions based on open RTOS which could be easily found in the market, each of which has the benefit according to the request. (memory footprint, device support, process scheduling hard real-time and so on). Concurrently, the operating system vendors offer a proprietary RTOS which guarantees that the developer can satisfy their application requirements and provide the support during the development phase. To adopt an open source RTOS, it can be a cost-effective solution from an economic standpoint and the innovation in the development, but it may be uneconomical if the development are complex and has unsufficient support for the software development and maintenance.

The proprietary solution is more expensive, but could be beneficial if the development support offered by the RTOS manufacturer is valid and efficient. Whatever the choice is, a careful analysis of RTOS, from the functional specification to the specific application, must have a throughout consideration, including the scalability and upgradeability, as well as those functional requirements.

Embedded systems have many tasks to be performed, each with its own deadline and can be defined only in its use; during the design phase, there are several requirements that must be considered:

- Functional requirements,
- Time requirements,
- Reliability requirements.

The timing requirements define the rigor of completion which can be, for example, the control of a loop or the answer to a user interface.

By the time, the functional complexity of embedded systems has grown exponentially and the real-time requirements have become increasingly important, as a consequence of the integration in the real-time operating system has become a requirement in the majority applications.

A real-time system is a system where the correctness of the computations rely on its logical correctness and the time instant in which the result is produced. If the constraints of the system are not met each other, the target will be missed.

There are two types of real-time systems (Fig. 6.1): reactive and embedded. Reactive real-time systems have constant interaction with the environment (such as a pilot for the airplane control). An embedded real-time system, however, is used to control a specialized hardware that is installed in a larger system (such as a microprocessor which controls the fuel-air mixture for automobiles).

The real-time embedded systems offer the outdoor control through sensors, actuators, and other input-output interfaces. Applications and examples of real-time systems are ubiquitous, can be found as parts of our activities in the areas of health infrastructure, educational and commercial purposes; includes the following systems:

- Vehicles for cars, airplanes, ships and railways;
- Traffic control for highways, airspace;
- Process control for power plants, chemical;
- Military use;
- Robot;
- Telephone communications and satellite;
- Home Systems.

The real-time systems must meet time constraints and the limited responses otherwise will suffer from the serious consequences. If the consequences consist of a reduction in performance, the system is defined as a soft real time (for example, the adjustment of the time of a computer on the network); constraints on time response are less stringent. If the consequences are about the system failure, then it is

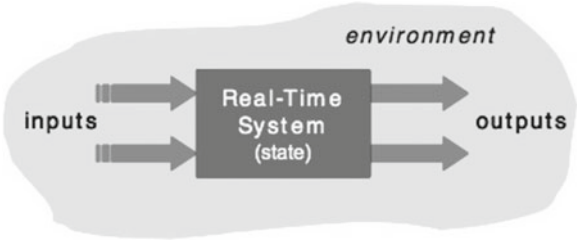


Fig. 6.1 Real-time systems

referred to a hard real-time system (e.g. intervention of the management of patients in hospitals); constraints on response times are stringent, otherwise the system will be considered useless or even dangerous.

Examples of real-time systems are included:

- Software for cruise missiles;
- Defence Systems;
- Telecommunication Systems;
- Control Automotive;
- Signal Processing Systems;
- Radar systems;
- Satellite systems;
- Electric Utilities.

The tasks of a real-time system can be divided into periodic and aperiodic. The tasks performed in each time unit are called periodic, otherwise, it's aperiodic. Real-time events belong to three categories: asynchronous, synchronous and isochronous. Asynchronous events are completely unpredictable. For example, the case in which a user makes a phone call. Synchronous events are predictable and occur regularly within a specified period of time. For example, the audio and video in a film are in a synchronous manner. The isochrone is a sub-class of asynchronous.

In conventional operating systems, processes of scheduling algorithms optimize the overall system performance; in real time systems, however, the scheduling algorithms are designed to optimize the time response in the case of soft real-time, and to ensure compliance with the deadlines in hard real-time systems.

The possible requirements of a real-time system (RTOS) are the following:

- Timing predictability of the system or all the services of the operating system must have an upper bound on the running time.
- The operating system must manage timing and scheduling or provide the precise time and ensure the deadlines.
- Configurable.

6.3 Examples of Real-Time Embedded Systems

Some operating systems can work in real-time with adequate hardware architectures, such as BeRTOS and LynxOS, architecture-based on Linux, Windows CE, Android.

Common features of the main RTOS are:

- Correspondence with the standards;
- Modularity and Scalability: the kernel has reduced in size with configurable functionality;
- Size of the Code;
- Speed and Efficiency;

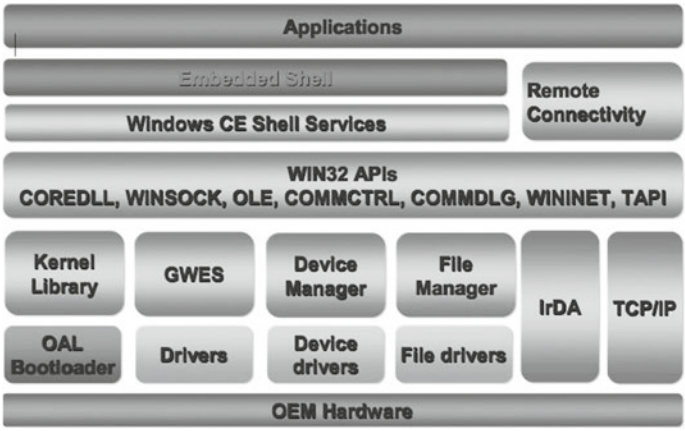


Fig. 6.2 Windows CE architecture

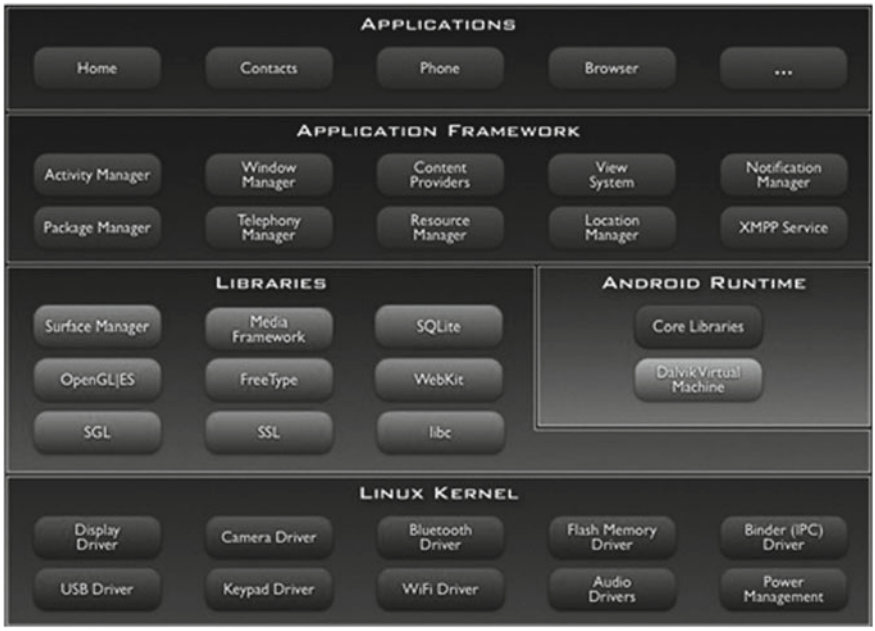


Fig. 6.3 Android architecture

- Interrupt;
- Memory management: ability to use virtual memory and protection of the kernel address space.

BeRTOS is an open source operating system, with a special closed code. Thanks to its modular design, it allows to work on multiple architectures: 8-bit microcontrollers such as the Atmel ARM microcontrollers.

Windows Embedded Compact (Fig. 6.2), also known as Windows CE, is an operating system developed by Microsoft for portable devices. It comes from Windows with a different kernel.

LynxOS RTOS is a real time Unix system developed by LynuxWorks. It offers full compliance with Posix and Linux. Used in the embedded systems, which are specially made for the applications for avionics, aerospace, military and industrial process control.

The LynxOS system is designed for hard real-time performance. Predictable response times are ensured in the presence of input-output devices as a result of the threading models in single kernel, which allows interrupting the routines to be extremely short and fast.

Android (Fig. 6.3) is a common operating system in the smartphones and tablets, based on Linux architecture. Its characteristics of open source and Linux kernel make it suitable for embedded systems [2, 5–7].

6.4 Scheduling

The response time and the execution constraints of individual processes are the key points of each embedded system, in this way, it is important to consider the heart of an RTOS: the Scheduler [2, 5–7].

There are multiple types of scheduler, but for some RTOS systems, do not reveal their feasibility:

- Round Robin Preemptive: each task is positioned into a queue and waits for its 'timeslice'.
- Non-preemptive priority: real-time processes are high-priority and carried out as soon as the currently being executed process.
- Preemption points, priority: the intervals are created when a running process may be discontinued if a high priority task needs to be run.
- Immediate preemption: the RT(Real-Time) requires an immediately operation, interruption of any running processes and respect of the most critical constraints of the systems.

One of the main factors that influences the choice of scheduler types is the schedulability analysis, which can be static or dynamic and is implemented by several classes of algorithms:

- Static-table-driven: using static analysis that allows to determine at runtime when a process must begin the execution. It's a flexible approach and not be changed in the requirements of a new model of scheduling.

- Static-Priority-driven: static analysis allows to assign priorities to tasks according to time constraints.
- Dynamic planned-based: a task is executed only if it is possible to avoid the time constraints.
- Dynamic BestEffort: simple and widely used; with the arrival of a task, the system assigns a priority based on its characteristics and eliminates the processes with violated timing constraints [4].

6.5 Scheduler Based on Deadline

The deadline scheduler based on the use of different key informations: reading time start or complete the deadline, processing time, necessary resources and possible substructure of the process.

The Earliest Deadline First (EDF) scheduler runs the task with nearest deadline, which will have the responsibility to leave the execution at the end of its critical section.

This technique allows to achieve better goals with a priority simple scheduler, but needs more information and more complex scheduling [2, 5–7].

Some introduction to other schedulers are as following:

- Cyclic executive: there is a supervisory program and tasks performed on the basis of what has been built into the design, executing a sequence of operations in a cyclic manner. It's a simple model but not dynamic.
- Deterministic: it provides methods to build scheduler in which the choice of the task to be scheduled at all known times.
- Capacity-Based: it requires information such as the amount of execution time in request for a task, which is calculated on the basis of whether it is possible to run the task on time. Each task must be periodic and the priority assigned to the task is higher (rate monotonic).
- Dynamic Priority: the priority of the processes is dynamic, considering the current state of the system to decide which tasks will be executed. The scheduler classics of this type is EDF.
- Imprecise results: each task is divided into optional/mandatory subtasks, the scheduler will ensure that the mandatory sub-processes are completed within the constraints, while the rest of the CPU time will be used for the execution of the sub-options.

6.6 RTOS for Multicore

One of the most important reasons for the choice of a multicore RTOS is the evolution of the computing platforms. The characterization of multi-core, combined with

the systems-on-chip, is particularly made for the complex systems and embedded computing. QNX Neutrino is an effective microkernel thanks to its modularity and adaptability for multicore architectures. Its Bmp technology (Bound Multi-Processing) eliminates the risk of migration which allows the developer to decide exactly where it will turn on the process and thread. Neutrino is a microkernel that ensures effective execution in a safe manner for each driver, application, protocol stack and file system.

In this way, each component can fail and run automatically without affecting other components or kernels. This is a peculiarity of QNX Neutrino that guarantees a high level control and the fault-related recovery [2, 5–7].

6.7 RTOS and Application Specific Processors

One of the most important connotations of embedded systems is the real-time performance and specificity of the processing, which more and more are oriented to manage the processes and information coming from the external devices. Generally speaking, Digital Signal Processor is the Application Specific Processors; in addition to the emerging technology of multimedia processors, they are new computing platforms for embedded systems. OSEck (OSE Compact Kernel) of Aeneas, the DSP-optimized version of the OSE RTOS, occupies only 8 kbytes of memory and is fully preempitive, event-driven, capable of producing real-time responses and able to detect and handle errors. These requirements are for hard real-time embedded applications, typical DSP applications and specific processors application.

OSEck uses a programming model for high-level message passing, which simplifies the decomposition of complex applications in simple concurrent processes, all communicating are through direct messages at high speed [2, 5–7].

6.8 An RTOS for Complex Systems

The complex embedded systems require absolutely determinism in process management and hardware devices. LynxOS RTOS is a real-time complex systems, and guarantee the absolute determinism (hard real-time). This means, response to the requests of the processes within a known time interval. The predictable response is ensured by the threading model that uses short and quick interruption handling routine. Among the features of LinuxOS, they are supported with memory up to 2 Gbytes and multiple applications with multiple switching devices [2, 5–7].

6.9 An RTOS Customizable

Freescale MQX RTOS is a configurable system which allows the developer to get a custom version according to its target. This RTOS allows to configure and balance the code to reach the required performance. MQX is tightly integrated with the computing platform for ColdFire which is available for the device drivers in common used. The integration with CodeWarrior development tools also allow to have very efficient debugging and profiling features. The density of MQX code is such as to obtain, after the configuration, an compact RTOS particularly for efficient, fast and small embedded system. It can be configured by occupying just 12 kbytes of ROM and 2.5 K of RAM on ColdFire V2, including the kernel, 2 application tasks, a traffic light LW, the stack for breaks and memory management. The real-time performance are guaranteed from the approach based on the priority and preemptive multithreading. This allows the high priority thread to be executed by the deadline in a consistent manner, regardless other threads are competing for the CPU.

Other features, such as fast boot, makes this particularly efficient on the complex powerful computing platforms [2, 5–7].

6.10 Interrupt

A microprocessor runs in more or less programs, transfers and receives data from external devices such as the monitor, keyboard, printer, disk drives and so on.

In addition, during the execution of the programs, there is also a continuous data stream from the CPU to the memory and vice-versa. To make the communication between the CPU and the input and output (I/O) devices, special techniques have been developed, that allow the hardware to exchange information with the microprocessor [2, 5–7]. The techniques used to make the data exchange between the CPU and peripherals, can be classified into three categories:

1. Polling: it is the simplest technique for the exchange of information between the microprocessor and peripheral devices.
2. Interruption: the technique is most widely used and it is of great importance in the data transferring between the CPU and peripherals. The interruption is maskable said when using dedicated instructions and the programmer can disable an interrupted signal, non- maskable when the programmer can not disable the interruption.
3. DMA (direct memory access): it is the technique that temporarily disables the CPU to allow the data exchange between memory and peripheral I/O or between memory and memory. To use this technique of DMA controller, which regulates the data flow from the device to memory and vice versa, is needed. With the DMA, it is possible to transfer data blocks at high speed.

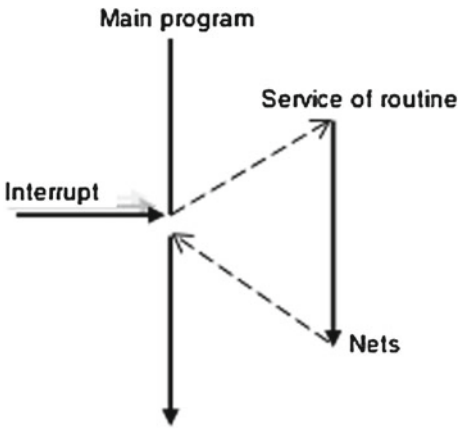


Fig. 6.4 Main program of interrupt

An interrupt of an event is sent to the CPU and stops the programming code flow. in fact it is used to perform operations in correspondence of events that are not temporally predictable, such as the timer overflow.

When the CPU receives an interrupt, it will suspend the current process, save the values of the registers and all the variables that describe the state of the running program, complete the execution of the current instruction and then execute the subroutine that corresponds to the interrupt handler. There may be, however, privileged instructions which do not allow the start of an interrupt to their conclusion, but only after the next instruction (Instruction boundary).

At each interrupt is assigned a specific address jump, 'interrupt vector', where the interrupt handler execution or Interrupt Service Routine is started. At the end of the code for the interrupt subroutine, the specific instruction NETWORKS (Return from Interrupt) is used to return to the execution of the program at the point where it was interrupted; In fact, once the function is required, the CPU fetches the old values of registers and resumes the program in execution that was interrupted, as shown in Fig. 6.4. In the memory of the microcontroller or microprocessor there is a table (Interrupt Vector Table) which contains, for each interrupt code, the relevant Interrupt Service Routine address.

6.10.1 Classification

Basically, the interrupts can be grouped into three classes:

- External Interruptions (external interrupt or device interrupts) are not caused by any instructions of the running program, but they are generated externally, for example to handle the I/O. They occur asynchronously to the execution of the program.

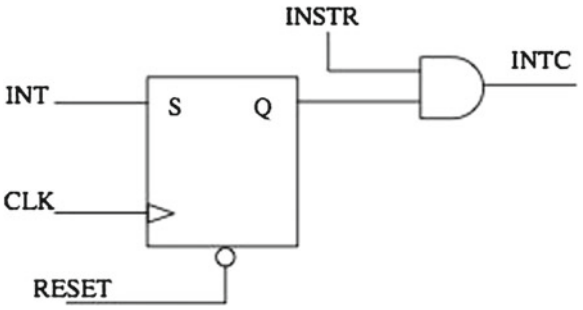


Fig. 6.5 Request of unconditioned interrupt

- Exceptions (exception conditions) are caused by abnormal situations detected during program execution, such as in an arithmetic overflow. They occur synchronously with the execution of the program, even if they are not predictable.
- Traps: interruptions are generated by specific instructions in the program which occur synchronously and predictable to the program execution. They are particular types of instructions which have the effect of bringing the system in the operation mode.

Interrupts are generally managed by a dedicated hardware and the Interrupt Controller, which collects the various interrupt signals, manages and enables them according to its priority.

Just a flip-flop and an AND gate, as shown in Fig. 6.5, enabled, disabled or resetted an interrupt request: if INSTR is a low logic level '0', the interrupt requests (INT) is disabled or masked, vice versa if INSTR is high logic level '1', the interrupt request is enabled.

Since it is necessary that an interrupt request does not unleash twice the execution of the subroutine, the recognition of interruption by following the instruction to reset the flip-flop in which the interrupt request is stored [2, 5–7].

The interruption technique is used to activate a specific subroutine only when there is a corresponding event according to its priority. In fact the management program of the interruption is performed by the programmer in a defined memory area, but it is never called by the main program (except for traps). In general, the subroutines are allocated to a memory area next to the main program and is achieved only on call.

The management of priorities is based on the easy following two rules:

1. A high priority interrupt can stop one more of low priority;
2. When there are two simultaneous interrupt requests, the one with higher priority is actived.

6.10.2 Management of Interrupt

In the moment in which the processor receives an interrupt request, is needed to determine what the device has to be generated it. There are three main systems for doing this:

- Multiple interrupt lines.
- Scan interrupt (polling).
- Vectoring of interrupts.

In multiple interrupt lines, each device is associated with an own flag for interrupt handling. This system is impractical for interrupt handling of many devices, but can be used in special-purpose technologies [2, 5–7].

The scan interrupt performs a device scan, questioning each of it for a possible confirmation of the interrupt request sent. When the device that generated the interrupt is questioned, will send a confirmation to the processor and then, it will start on the ISR. Setting the scanning sequence is also possible to set the priority.

The vectorization makes use of an integrated circuit named Programmable Interrupt Controller (PIC) that has a number of input lines of the interrupt requests (IRQ).

When a request is received, the PIC is responsible to send an interrupt request to the processor. Then it is deposited in the data bus index of the interrupt vector on the ISR that handles the interaction with the device.

The vectors are contained in a table located at a specific location in memory. Each vector contains the data and allows to the processor to determine the memory address of the ISR for the device that has generated the interrupt and then handle the I/O request.

6.11 Linux

Linux is a general purpose operating system, characterized by a non-preemtable kernel: the goal is to give for each process a fair portion of the system resources (CPU, memory, use of peripheral devices, etc. ...) without being able to stop the activities of the kernel (such as the Interrupt Service Routines, ISR). The design given by Linus Torvalds to its software is away from the real-time factor: if you would like to use Linux for this scenario type, some structural changes have to be made in order to adapt the architecture to a different behavior regarding the management of the task execution flow.

As an example, let us refer precisely that the kernel is non-preemptable: It is necessary to run a real-time task while the kernel is executing in its internal routine (such as interrupt management). The real-time tasks must necessarily wait for the routine ended before to re-use the CPU.

This is unacceptable in a real-time system. However, it makes sense to include the changes to adapt to the real-time Linux for various reasons [2, 5–7]:

- Linux is a high-level and open source operating system;
- Well documented in all its parts;
- Born in the university and already rooted in various research contexts;
- Compatible with the standard to the software development (POSIX, etc. ...);
- The monolithic structure makes it suitable in the scenarios embedded;

6.11.1 Problems

System memory in Linux can be divided into two areas: user space and kernel space. The user space is where the user can run applications (composed of processes that do not belong to the kernel as word processors, games, etc. ...) and is organized on a time-division mechanism: each task is assigned to a time quantum in relation to the task's priority.

Kernel space, instead, is where the kernel (i.e. the core of the operating system) executes (runs) and provides its services.

In addition to time-division mechanism, Linux is equipped with two additional scheduling algorithms to use in real-time applications: the algorithm FIFO and Round Robin.

Round-robin (RR): is one of the algorithms employed by process and network schedulers in computing. In order to schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each task a time quantum (allowance of CPU time), and interrupting the task if it is not completed by then.

FIFO method: in this way the tasks are scheduled in an ordinate sequential where the requests are executed into kernel (non-preemptable). If during the execution of the hardware, an interrupt is verified, the kernel is forced to stop the FIFO queue execution and starts the Interrupt Service Routine [2, 5–7].

This is not acceptable in hard real-time. Then, the system does not proceed properly, although the algorithm is valid for the scenario for consideration. It is possible to think of the use of this scheduling mechanism only for some soft real-time scenarios.

Unfortunately, due to the monolithic kernel structure, the non-interruptible sections are particularly long: in some cases on an Intel Pentium 3, are calculated up to 100 ms of time, which is too large for soft real-time scenarios as performance degradation for a single process. The delays described above, is called scheduling latency that represented the fundamental limitations to Linux in the real-time field. The scheduling latency is the time between the last instruction of the user's interrupt handler and the execution of the first instruction of a driver thread.

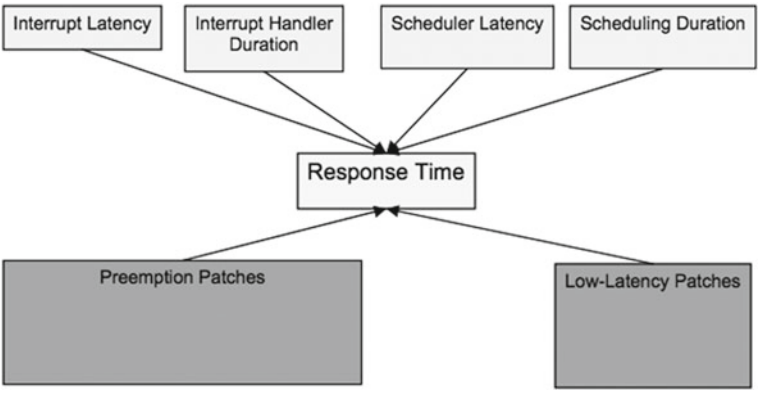


Fig. 6.6 Time response factors for the Linux kernel

6.11.2 Real Time Linux

The Fig. 6.6 summarizes the factors that are the most influence on the time response of the Linux kernel, and then introduce latency in the system.

Some designers have developed kernel patches, which are available and easy to find on the network, to reduce this effect and can be used in real-time by reducing the indeterminism.

The patches are two, both aimed at reducing the scheduling latency. The Preemption Patch going to change the operating frequency of the kernel by reducing the period of the schedule function. Latency Patch adopts a different philosophy: the goals is to reduce the latency by inserting so-called preemption points in the non-interruptible source code of the kernel so that the system can have a better reactivity [2, 5–7].

Despite the clear limits of real-time Linux, various technologies have been developed to provide the system with software tools that provide for real-time performance and in some cases have been obtained very satisfactory results. Two main routes to consider:

- Replacement of the Linux kernel with a totally different preemptable kernel;
- Co-Linux kernel, a kernel (known in the literature as the Resource Kernel) can provide the real-time performance to the system.

The first solution, shown in Fig. 6.7, provides both real-time tasks managed by the same scheduler and, therefore, able to recognize the tasks in different types and make appropriate choices about their execution.

The second architectural, shown in Fig. 6.8, has some substantial differences. First, the system management will be made by Resource Kernel. Another fundamental difference is that the task scheduler must be managed in different way depending on whether or not the real-time tasks.

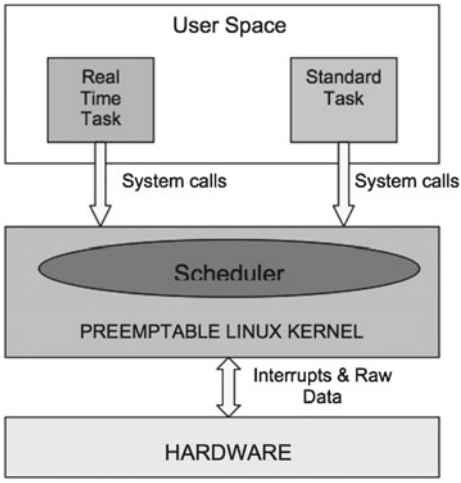


Fig. 6.7 Architecture with preemptable kernel

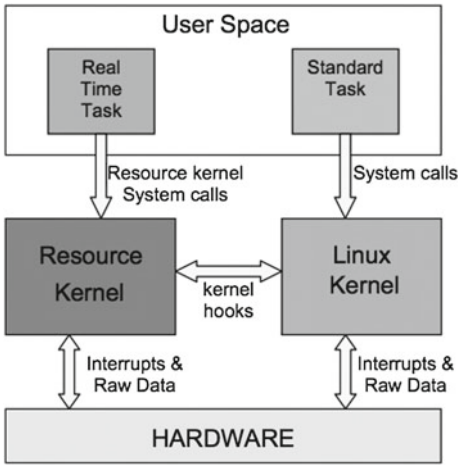


Fig. 6.8 Architecture with resource kernel

Consequently, the programmer must be aware of the fact that there are different system calls for real-time tasks compared to those of common tasks and coding standards (like POSIX) that may not be presented for the real-time: everything depends on the implementation of the Resource Kernel selected [2, 5–7].

References

1. Razavi, B. (2008). *Fundamentals of microelectronics*. New York: Wiley.
2. Ganssle, J. (2008). *The art of designing embedded systems*. Oxford: Newnes.
3. Di Paolo Emilio, M. (2013). *Data acquisition system from fundamentals to applied design*. New York: Springer.

4. Valdes-Perez, F. E., Pallas-Areny, R. (2009). Microncontrollers, fundamentals and applications with PIC CRC Press.
5. Heath, S. (1997). *Embedded systems design*. Oxford: Newnes.
6. Vahid, F., & Givargis, T. (2002). *Embedded system design: A unified hardware/software introduction*. New York: Wiley.
7. Wilmshurst, T. (2008). *Designing embedded systems with PIC microcontrollers*. Oxford: Newnes.

Chapter 7
Design PCB for Embedded System

Abstract The design of a PCB is a very important task to realize electronic prototypes efficiently from both an operational point of view and commercial. Basically, in the embedded applications, the design of the PCB plays a key role. The circuits are embedded in various types and sizes, in relation to the type of microprocessor, component and operating system, above all, the complexity of the software can be various from a few hundred of bytes to several megabytes of code.

7.1 Materials for Printed Circuits

A printed circuit board is a set of copper tracks suitably drawn on an insulating support and used to connect the components that constitute the electronic circuit. The base material is formed by the copper sheet of appropriate dimensions and the insulating part that can be in different types according to the performance; the classic Bakelite is the insulating material which is cheaper and has less performance.

The PCB (Fig. 7.1) are made by assembling thin dielectric layers which are made with electrically conductive material:

- One or more rigid or flexible substrates act as insulation.
- Layers on copper levels conduct the electric connections.
- Holes conductors ('vias') cross-connect different layers.
- A painted surface is used to protect the slopes from oxidation and facilitate welding ('solder').
- Screen printing ('Silkscreen') is to mark the location of the components on the board.

For the choice of the dielectric material of the substrate, there are some factors needed to be considered.

Coefficient of Thermal Expansion (CTE): The coefficient of thermal expansion is the tendency of the material to change in volume in response to a variation of

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_7

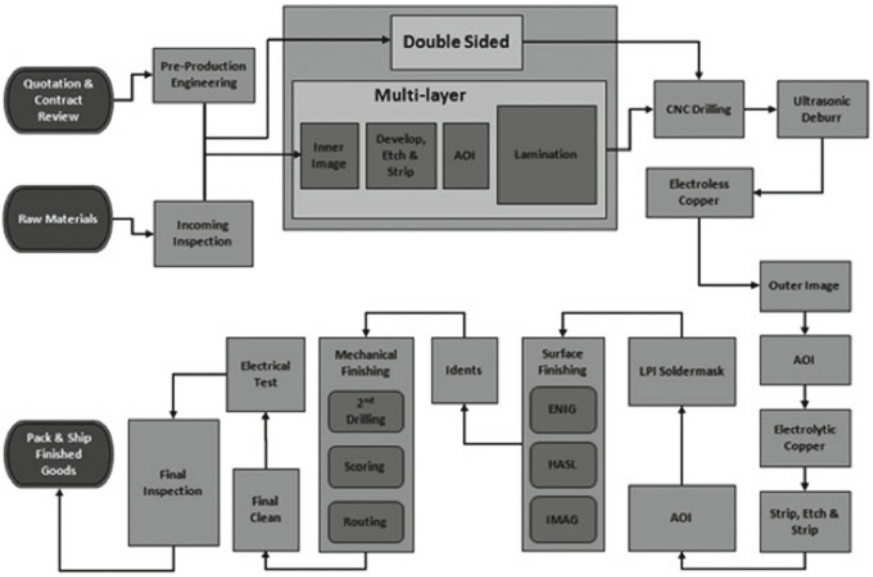


Fig. 7.1 PCB design flow

temperature. When a substance is heated, the particles begin to move more and thus usually maintain a higher average separation.

Glass Transition Temperature (T_g): The glass transition is a property of the amorphous portion only a semi-crystalline solid. The crystalline portion remains crystalline during the glass transition.

Thermal conductivity: The ratio between the heat flow and the temperature gradient. It represents the measure of the ability of a substance to conduct the heat, generally depends on the nature of the material itself but not on its shape.

Rigidity mechanics: The ability of a body to oppose an elastic deformation due to an applied force.

FR4 substrate (Fig. 7.2) is most commonly used to make circuit boards; formed from glass fibers and copper joined by an epoxy resin. It is the most common material used in electronics and mechanics, lightweight with high resistance to mechanical stress and good resistance to thermal shock of short duration.

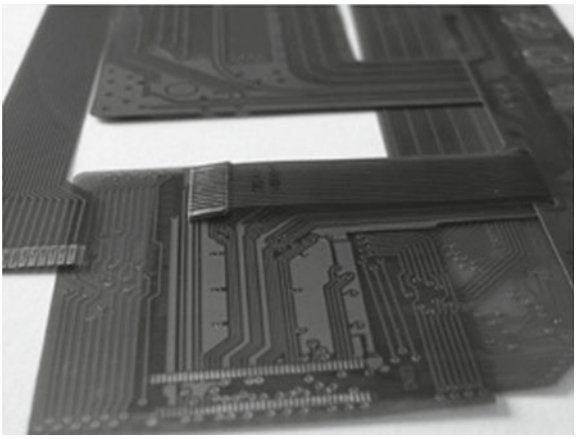
FR-4 is acceptable for signals up to about 2 GHz, depending on the application in used, in addition, for the production of insulating and structural components. If the same board of FR-4 delivers more high-frequency signals, it would cause power loss and interference increase. Other materials, provide superior electrical characteristics at higher frequencies.

Substrates commonly used are the following:

- Polymide/fiberglass;
- FR2;

Fig. 7.2 FR4 substrate

Fig. 7.3 Kapton



- KAPTON (Fig. 7.3): flexible and lightweight, used for specific applications (displays, keyboards).

The polyamides are known for its thermal stability, chemical resistance and excellent mechanical properties. The compounds are reinforced with glass fiber or graphite to a high flexural strength up to 345 MPa. Kapton (Fig. 7.3) is a polyamide film developed by DuPont which is enabled to remain stable for temperatures ranging from −269 °C to +400 °C. It's used as well as in flexible printed circuits even in the astronauts space and suits to ensure thermal protection.

For the realization of a PCB performing in different steps of manufacturing, particularly, the first phase of the set-up consists in the choice of materials, processes and requirements. The phase can be defined in the following points:

- It is called the quality and reliability of the card, with reference to the specifications.
- The materials are chosen for the core and the layers of materials referring to the IPC-4101.
- Stack-Up is defined layers (core stack-up or foil stack-up).

Other materials which may be used in on PCB are:

- CEM-1. It consists of two layers of woven glass fiber with a layer of paper in the middle; both of these materials are impregnated with an epoxy resin. It has the best electrical and physical characteristics compared to the FR-1 and FR-2.
- CEM-3. It consists of two layers of woven glass fiber, with a non-woven glass fiber in the middle, both impregnated with an epoxy resin.

Besides the choice of material, is fundamental importance of the choice of the PCB coating; coating is made through the special protective resins and consists in the assembling in the PCB with a resin film that crystallizes at a certain temperature. It forms a single body with the substrate and welded components, isolates them from the external elements.

The main goal is to obtain a barrier of insulation for electronic components, so that the PCB can run in any operating mode; in this way, the damage is limited due to mechanical stress, humidity and more.

The resins in used are in acrylic-based and silicone, which is selected according to the operating environment.

The main characteristics of the most common resins in used can be listed in the following:

- Epoxy resins:
  - Good moisture and temperature resistance;
  - Thermal shock resistance.
- Resins puliuretaniche:
  - Moisture and thermal shock resistance;
  - Low temperature resistance.
- Silicone resins:
  - Excellent electrical conductivity;
  - Non-economic.

Polyurethane resins are preferred when PCBs are formed by relatively delicate components such as ferrites that might be interfering RF signals. Silicone resins, however, are very expensive for its excellent electrical capacity and are used when the PCB operates at high temperatures (higher than 180 °C) [1–4].

7.2 Electrical Insulation on PCB

The electrical insulation is a condition in which between two points, with a different potential, there is a movement of continuous current. At the physical level, there is a shift of electrical charge from one point to another while the electrical power is exchanged by electromagnetic inductive or capacitive phenomena. The magnitude of the electric type that urges an insulating material, defined as stress dielectric, is the electric field E [V/m]. When the stress is too high , the dielectric could cause a temporary or permanent damage (depends on the material type) of the insulation that impairs the functionality of the machine to which the insulator shall be included.

It is defined as E_r dielectric strength [V/m] of a material the maximum value of the dielectric stress which may be applied without damage.

The dielectric strength (Fig. 7.4) is essentially a random parameter which must be done on a statistical basis. It also depends on several factors:

- Waveform and duration of the applied voltage;
- Geometry of the electrodes and insulator;
- Physic-chemical characteristics of the material;
- The presence of impurities in the material (moisture, gas, waste, etc.);
- Thermal and mechanical stresses applied to the material.

The electric field in an insulating material is determined by the applied voltage V and its geometry: $E = -\nabla V$ is proportional to the applied voltage, but in the homogeneous material, it does not determined by the permittivity of the material itself. The dimension of the dielectric insulating material consists in determining of the geometry so that the pressure does not cause the damage to the dielectric; this means that the size and shape of the material must be conformed with the dielectric stress that is lower than the dielectric strength: $E < E_r$.

Since the dielectric stress E is determined on the applied voltage V, it is obviously that the dimension must be done in relation to a well determined value of the applied voltage.

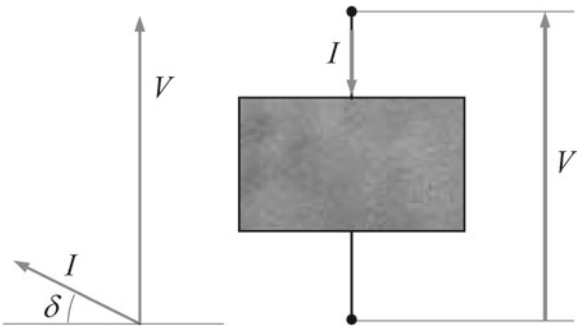


Fig. 7.4 Dielectric strength of insulation

It defines the level of insulation value of the applied voltage that determines the dimension of a dielectric insulating material.

The level of insulation is usually higher than the value of the nominal voltage, and this have the following characteristics:

- The size of the insulation must take consideration of the surges, long-lasting or transient that can stress the system under certain operating conditions or anomalies effect.
- In connection with the reliability is intended to provide to the equipment: more security and highest level of isolation.

The correct sizing of isolation is verified with tests in the laboratory by applying insulation test voltages which is provided by the insulation level; it usually indicates the test voltages that the machine or equipment must be ensured to remain undamaged.

A PCB (printed circuit board) must be isolated to ensure the protection level and, therefore.

FR-4 is preferred as one of the PCB material, economic alternative solution could be as synthetic resin bonded paper (SRBP, FR-1 , FR-2). Due to some of its physical-mechanical characteristics and the ability to keep data at high frequencies, flame and heat resistance, water absorption is less than other materials, hence FR-4 is widely used for the high-end construction in the industry, military and consumer electronic equipment. It is also compatible with the 'ultra-high isolation' (ultra high vacuum—UHV).

The realization of the tracks in the PCB must be complied with the certain protocols, particularly, for the printed circuits at high voltage and/or current is required more tracks width and isolation distance. The choice of sizing must not only follow certain rules but also specify the environmental techniques for determining the place where the PCB will be operated. For the evaluation of the maximum current, a parameter to consider is the heating curves. For thicknesses of 35 μm whiskers graph allows to give an assessment of the width of the track as a function of the temperature of the track itself. To determine the surface's temperature, it must be added the temperature indicated in the graph of Fig. 7.5 with that of the environment. The current display is in average effective value (RMS). Consider the passage of a current of 10 A to limit overheating at 10 °C: the width of the track, in according to the graph of Fig. 7.5, is 9 mm. In the same track can pass about 20 °C causing an overheating of 30 °C.

Similarly, it is possible to use a 2.5 mm track if there is no problem with the high temperatures. High current requires the high copper thickness (70 μm) of circuits and use of large copper areas; it is important to note that a surface connected to a copper track allows to lower the temperature.

Sometimes the problem can be traced back to the resistance of the track (and therefore the voltage drop). To calculate this resistance may be used the classic law:

$$R = \frac{\rho L}{A} \tag{7.1}$$

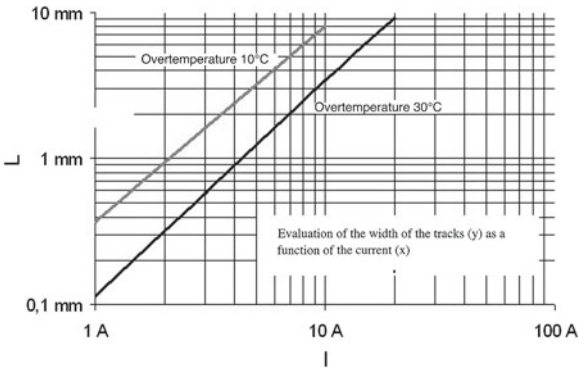


Fig. 7.5 Slopes size evaluation

the resistivity of copper at 25 °C is about 0.018 μΩ meter. A track with a 10 cm (L = 0.1 m), by the width of 1 mm and the thickness of 35 μm has a resistance of about 0.05 Ω. The parameters that allow an evaluation of the insulation distances are known as clearance and creep age. The first measured in air, the second measured in the following periphery of the insulating surface. A wide variety of environment (humidity, pressure, etc.) is very difficult to make an accurate assessment [3–6].

7.3 Routing PCB

The problem of routing for PCB is divided into two main phases: the global routing and detailed routing. The global routing includes the study on the optimal placement of components on the base and the identification of the paths that approximate the tracks will have to make from one component to another. This stage is very important because it allows splitting the base in various areas on which will go then to act the detailed routing.

The task is delicate because it is necessary to evaluate accurately the density of tracks and obstacles in each area so that it is then possible to complete the routing. The detailed routing is concerned to trace the connections between the components in each area identified from the global routing technology and respecting the constraints imposed on the project and optimizing some characteristics of the path set by the designer. The constraints to be respected in the tracking of the slopes can be the following: minimum distances of track, minimum width of the tracks, directions binding of the slopes (usually vertical and horizontal). The main goal of detailed routing is not to minimize the length of the tracks, once is established that there is a path, it could be more significant to find the path that will lead to less difficulty routing of the tracks which have not yet been traced. Or the attention may be paid to minimize the number of passages from one face to the other in the case of a double

sided PCB. Or to minimize the total area occupied by the circuit. As a final example, is possible to think how to minimize the critical path of some tracks in order to minimize the delay of the signal that travels [3–6].

7.4 PCB Embedded

The design of the PCB is limited by its application from its 'comfort'. Portable devices, as mobile phones, are a perfect example, they have to fit in the hand, approach the ear to hear and equipped with a suitable size for easy use. These represent the minimum standards that have been changed since the introduction of the first GSM mobile phone. Still, consumers see higher capacity in the new handsets with the same size requirements.

There is an increase of active and passive electronic components, however, the space between these components are even more limited for electronic circuits design; then PCBs become denser with additional components, tracks and vias.

The number of discrete passive components is about 70–80 % of the total number and continue to grow much more. While the active components can be made in chips, the ideal space for placement the discrete passive components become increasingly difficult to obtain. There are some considerations of which need to take into account:

- Reduce the length of the critical paths;
- Analog components separated physically from digital;
- Power components physically separated from other;
- Orientation of the components agrees with that of the slopes;
- Distribution and sizing of capacitors and filters for noise reduction (low and high frequency) or external coupling.

A component can be an embedded PCB with integrated active and passive components in a resin substrate. The design of a PCB with embedded monolithic ceramic capacitors is shown in Fig. 7.6. This scheme allows, as mentioned previously, an expansion of the structure components PCB and reducing the physical size. To facilitate the operation of these PCBs, a high quality power supply must be provided in order to absorb fluctuations in load and eliminate the noise during operation. For this reason, it is essential to reduce the inductance component by placing the passive components, such as a capacitor which is close to an IC in order to reduce the length of connection and to avoid unacceptable levels of noise that cause timing errors.

The use of embedded passive components allows using the use of the circuit at high frequencies. The linearity of the signals through these components also help reduce the inductance to the surface and improve the electrical efficiency of the PCB. The reliability of the PCB would be improved through the reduction of the total number of welding joints. A welding joint is a point or edge where two or more pieces of metal or plastic are joined together. They are formed by welding two or more work pieces (metal or plastic) according to a particular geometry.

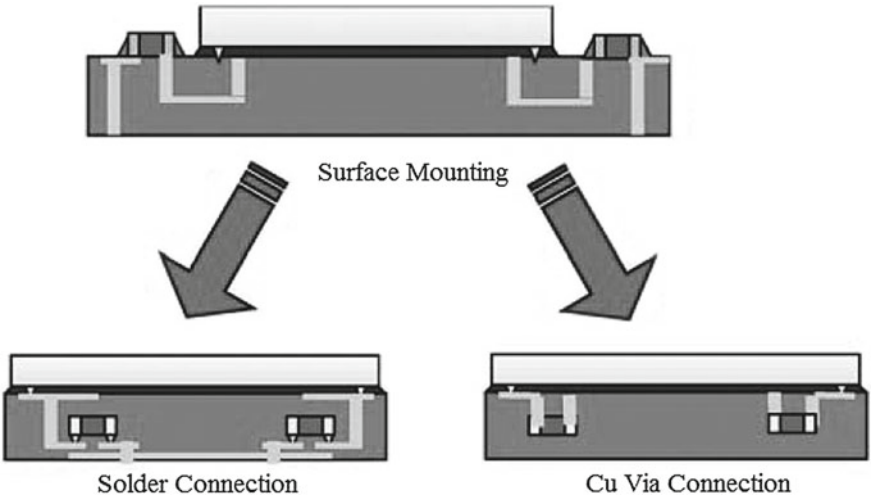


Fig. 7.6 Schematic of an embedded PCB

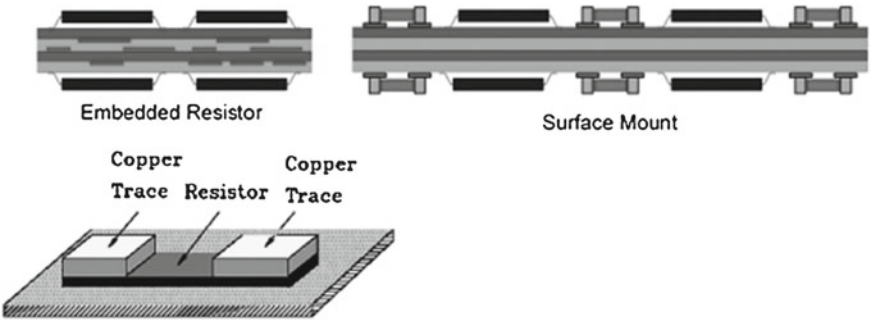


Fig. 7.7 Embedded resistor

A resistor embedded resides on a single level, which can be any physical layer. It has two copper pads with resistive material between the electrodes. The shape of the resistance material may be a simple rectangle, or a shape shown in Fig. 7.7. In all cases, the resistance material overlaps the copper pads [3–6].

In general, the capacitors represent the highest percentage of components on a PCB. Today there are two techniques for the manufacture of embedded capacitors (Fig. 7.8): through the use of dielectric films and a process additive. The later is a screening process which is created directly on a layer and is used in MCM and hybrids.

Moreover, Embedded Inductors are realized by thin lines of spiral shape. The combinations of line width, number of turns, shape and length define the inductance. Figure 7.9 shows spiral lines of copper connected through a pad.

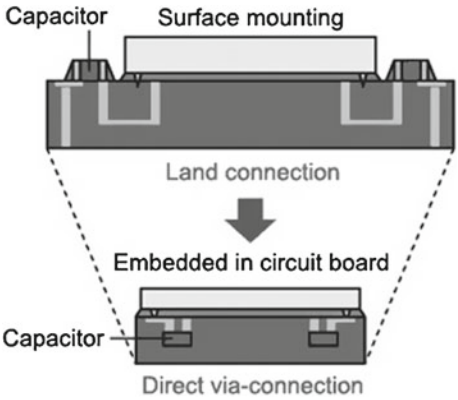


Fig. 7.8 Embedded capacitors

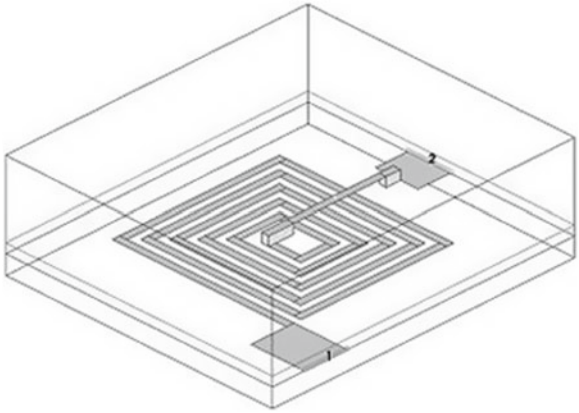


Fig. 7.9 Embedded inductors

7.4.1 Design Guidelines

One of the most important points to understand the use of integrated passive components embedded is a manufacturing process but not an assembly process. Considering in the design phase, this provides a great flexibility allowing to designers of PCBs to deal with the growing challenges of component density. Of course, such flexibility can be realized only if the Tools fully support embedded components. From the point of view of the design flow, the synthesis of the component must take place in the phases of positioning and interconnection of PCB design. This provides the optimal control of the design on the size of the component and the XY positioning on the layer during the definition of the critical aspects of the design. A summary of the appropriate component provides designers different physical options. Thus, two instances of resistance from 10 kΩ, for example, can be represented in a completely

different way in a design for the best suit the design density. Interactive tools allow the designers to have an automatic synthesis forms and individual values or in groups selected in order to optimize the design density. Embedded discrete active components have different needs. They can be placed in a cavity exposed or embedded within the layers of the substrate. Compared to passive ones, have a much greater thickness, both for the physical component and assembly.

Moreover, the final stage of PCB design is the post-processing - the generation of outputs. This activity is typically done by the PCB designer. Design tool should provide Boolean AND, OR and XOR, which includes materials to determine what is included on the outputs of the plotter. These are used to create masks of appropriate materials and manufacturing processes. Additional, outputs must support laser trimmer, XY positions, layer pads and wire bonds. All these output capabilities are needed for each substrate.

Embedded components provide the opportunity for greater functionality, performance and density of PCBs at lower costs. However, these benefits can only be realized with the use of design tools, which includes and embraces the unique needs of the components. Tools for designing PCB must allow the flexibility to adopt the component at any point in the drawing so that it can be accomplished the best PCBs in term of density [3–6].

References

1. Razavi, B. (2008). *Fundamentals of microelectronics*. New York: Wiley.
2. Kang, S. M., & Leblebici, Y. (1998). *CMOS digital integrated circuits*. New York: McGrawHill.
3. Di Paolo Emilio, M. (2013). *Data acquisition system from fundamentals to applied design*. New York: Springer.
4. Valdes-Perez, F. E., & Pallas-Areny, R. (2009). *Microncontrollers, fundamentals and applications with PIC*. Boca Raton: CRC Press.
5. Harper, A. (2000). *High performance printed circuit boards*. New York: McGrawHill.
6. Thierauf, S. C. (2004). *High-speed circuit board signal integrity*. Boston: Artech.

Chapter 8
Features of High Speed Data Acquisition and Control System

Abstract The integrated control systems represent one of most important modern and developed areas of electronics. The industry automation and intelligent systems have a variety of application in many fields from biomedical to the automotive. Data acquisition systems have evolved over time from electromechanical recorders which are composed typically of one to four channels, to all-electronic systems and are capable of measuring hundreds of variables simultaneously. PCB layout is one of the last steps in the design process and also one of the most critical. High-speed circuit performance is heavily dependent on the layout. A high-performance design can be rendered useless due to a poor or sloppy layout.

8.1 Data Acquisition System

Data Acquisition Systems (DAQ) are the main instruments used in laboratory research from scientists and engineers; in particular, for test and measurement, automation and so on. Typically, DAQ systems are general-purpose data acquisition instruments that are well suited for measuring voltage or current signals. However, many sensors and transducers output signals must be conditioned before that a board can acquire and transform in digital the signal. The basic elements of DAQ are shown in the Fig. 8.1 and are:

- Sensors and Transducers
- Field Wiring
- Signal Conditioning
- Data Acquisition Hardware
- Data Acquisition Software
- PC (with operating system)

Transducers can be used to detect a wide range of different physical phenomena such as movement, electrical signals, radiant energy, thermal, magnetic or mechanical energy etc. They are used to convert one kind of energy into another kind. The type of input or output of the transducer used, depends on the type of signal detected or process controlled; in other ways, we can define a transducer as a device that converts one physical phenomena into another one. Devices with input function are

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_8

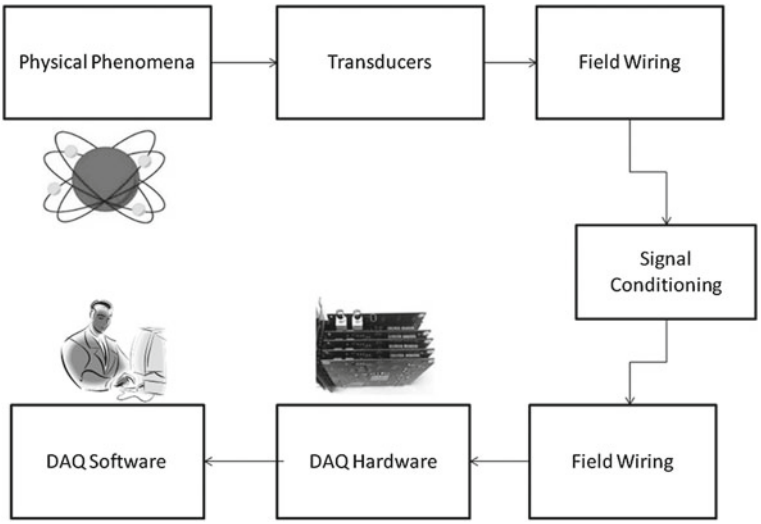


Fig. 8.1 Functional diagram of a data acquisition system

called Sensors because they detected a physical event that changes according to some events as for example heat or force. Instead, device with output function are called Actuators and are used in Control System to monitor and compare the value of external devices. Sensors and Transducers belong to category of Transducers.

There are many different types of transducers; each transducer has input and output characteristics and the choice depends on the goal of your system; for example from type of signal must be detected and the control system used to manage it (see Table 8.1).

Sensors produce in output a proportional voltage or current signal in according to the variation of physical phenomena that are measuring. There are two types of sensors: active and passive. Active sensors require external power supply to work; instead, a passive sensors generate a signal in output without external power supply. Signal conditioning consists in manage an analog signal in order that it meets the requirements of the next electronics system for additional processing. Generally, in

Table 8.1 Common transducerss

Quantity being measured	Input device (sensor)	Output device (actuator)
Light level	Photodiode photo-transistor solar cell	Lamps-LED-fibre optics
Temperature	Thermistor–thermocouple	Heater–fan
Force/pressure	Pressure switch	Electromagnetic vibration
Position	Potentiometer–encoder	Motor
Speed	Tacho-generator	AC/DC motors
Sound	Carbon microphone	Buzzer–loudspeaker

various applications of control system there is a sensing stage (for example a sensor), conditioning stage and a processing stage. The conditioning stage can be built, for example, using operational amplifier to amplify the signal and, moreover, can include the filtering, converting, range matching, isolation and any other processes required to make sensor output suitable for processing stage. The processing stage manages the signal conditioned in other stages such as Analog-To-Digital Converter, micro controller and so on [1].

8.1.1 Data Acquisition Hardware

Data acquisition systems (DAQ) can be defined as a set of electronics system, which any of the following functions:

1. The input: processing and conversion to digital format using ADCs. The data is then transferred to a computer for display, storage and analysis.
2. The processing: conversion to analog format, using DACs. The analog control signals are used for controlling a system or process.
3. The input of digital signals, which contain information from a system or process.
4. The output of digital control signals.

In general DAQ hardware is the interfaces between the analog signal and a PC. It could be in the form of modules that can be connected to the computer via serial and USB port for example, or cards connected to slot in the mother board: for example PCI or PCI express.

Many Data Acquisition Systems, known as plug-in boards, are used in scientific application to acquire data and transfer it directly to computer memory. Transference of data can be done by parallel port, serial port, USB port, Ethernet port and so on [1].

Typically, DAQ plug-in boards (Fig. 8.2) are general-purpose data acquisition instruments that are well suited for measuring voltage or current signals or resistance that can include some form of signal conditioning [1].

Usually, application in real time of high frequency analog signals need a high speed DAQ with a dedicated plug-in processor such as a digital signal processing (DSP) board.

The main components of DAQ system is the Analog input (A/D) boards, it converts (Figs. 8.3 and 8.4) analog voltages from external signal sources into a digital signal, which can be read by the host computer. Moreover, the functional diagram of a typical DAQ system can be described of the following main components:

- Input multiplexer
- Input signal amplifier
- Sample and hold circuit
- A/D converter
- Memory (DMA)
- Timing system and filtering

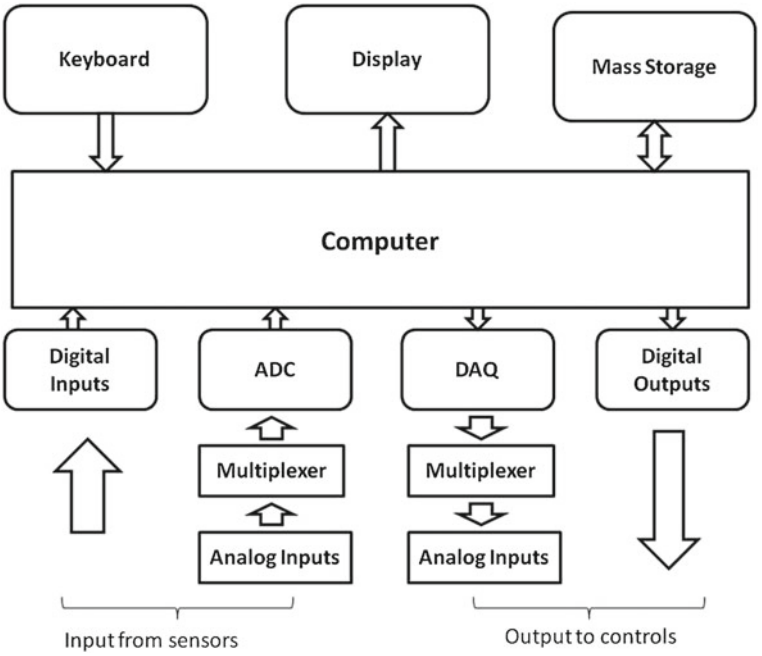


Fig. 8.2 Functional diagram of a basic data acquisition system

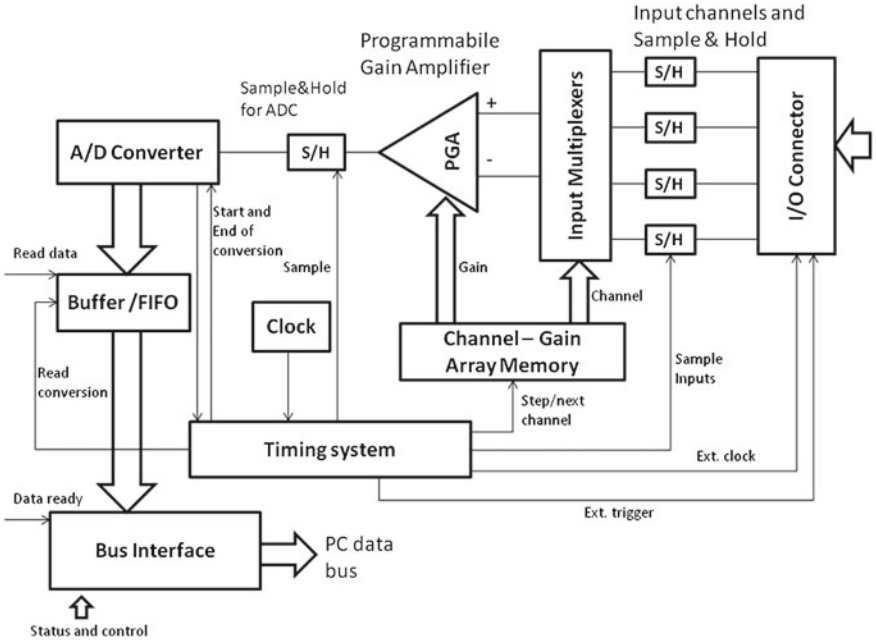


Fig. 8.3 Functional diagram of a generic A/D board

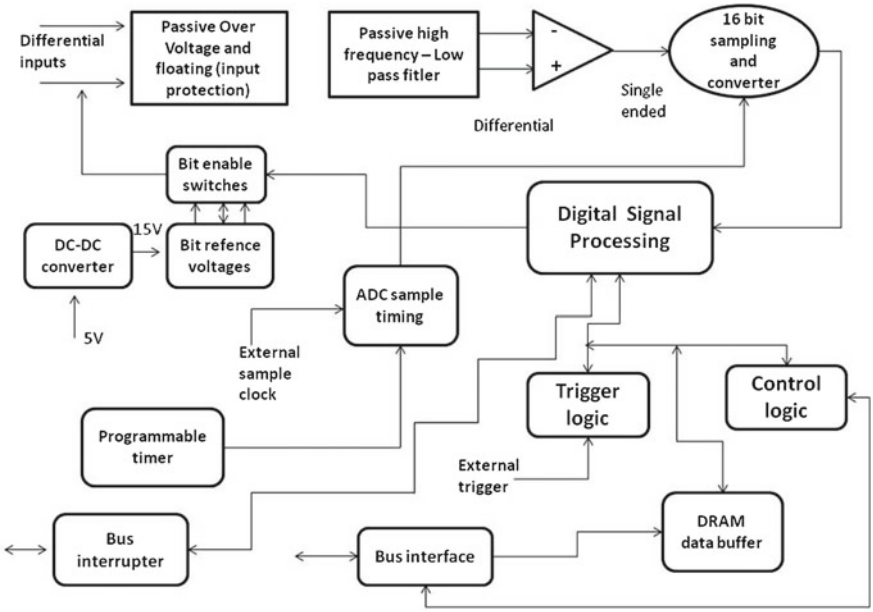


Fig. 8.4 Functional diagram of a typical DAQ system

- Bus interface
- Digital signal processing
- Microprocessor and/or field-programmable gate array

The performance of DAQ system can be affected from data transfer capacity of the board. Usually, all PCs are capable to program I/O and interrupt transfers.

Computers that have DMA channels can transfer data using much less CPU laden than those which without a DMA channel.

Without DMA, CPU is typically occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU initiates the transfer, executes other operations while the transfer is in progress, and receives an interrupt from the DMA controller when the operation is done. This feature is useful when the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, as for example disk drive controllers, graphics cards, network cards and sound cards [1].

8.1.2 Data Acquisition Software

DAQ software is the main component of the DAQ system, it is needed in order for the DAQ hardware to work with a PC. Data acquisition software (Fig. 8.5) can

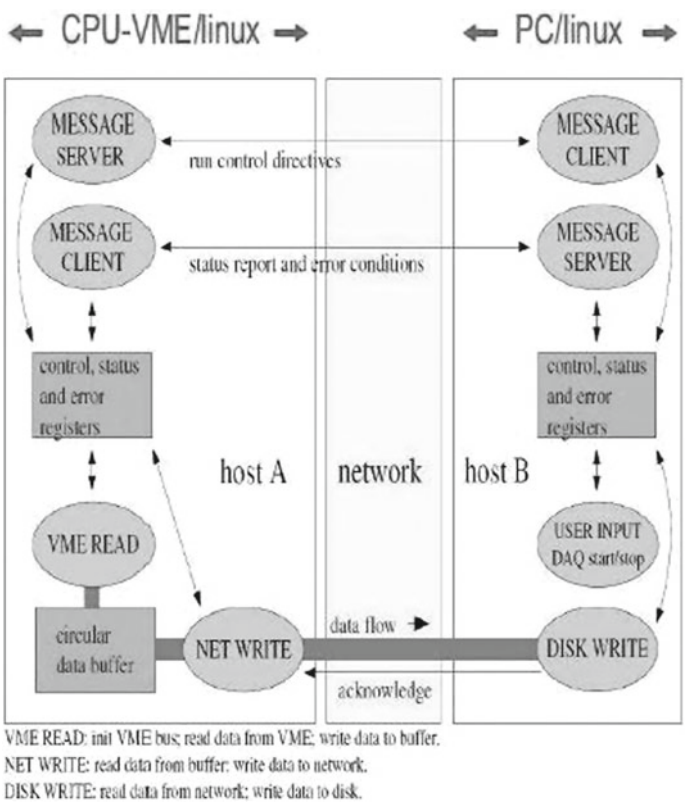


Fig. 8.5 Example for software architecture of a data acquisition system via VME bus

be written in a variety of languages (for example C language) and can be written for the particular application in design. Alternatively there are a number of proprietary data acquisition software packages that are available and these can be utilized (see National Instrument with Labview). Usually, DAQ software is composed of a text based user interface (TUI) consisting in an ASCII configuration file and and a graphic user interface (GUI) available by means, for example, of any Web browser. Both interfaces permit DAQ management and customization without the need of recompiling the sources, thus granting full acquisition control also to inexperienced programmers. The configuration file is written in a high-level language (meta language) and is easily modified by the operator. The GUI works at a higher level with respect to the ASCII configuration file and helps the operator in compiling the configuration file and in controlling the acquisition. The use of the Web interface does not require any knowledge of the configuration file sintax and avoids grammatical errors. It is up to the operator to choose the TUI or the GUI when modifying the DAQ setup [1].

8.2 High Speed PCB Layout

The PCB consists of several layers of metal and insulator. Copper trace is utilized in the connections between components, and the shape of this trace represents an important aspect of a PCB, which it determines the characteristic of inductance, capacitance and impedance. Resistance is generally ignored in the mostly designs which does not carry more than several mA of current and the results can often be negligible (Figs. 8.6 and 8.7).

Typically, Copper planes are used when the power planes and ground planes are requested. Planes make an excellent high frequency capacitor and can often be utilized for high frequency bypassing in complement with traditional capacitors. Usually, a solid ground plane is preferred over a grid plane that minimizes inductance to the absolute minimum value which is a desirable trait for high speed signals (Analog and Digital signals). With this plane, it can cause the capacitance problems to the sensitive nodes of the circuit. Be aware of all attributes of the circuit and do not blindly use planes everywhere. A good thermal factor can do the solid plane as a heat sink to keep thermal levels of all devices minimized; on the other side, temperature sensitive components may avoid to be placed near to the ground plane due to this heat spreading [2–4].

Vias are used in PCB when there is high density of interconnections (i.e. BGA packages) to simplify trace routing around components. As described previously,

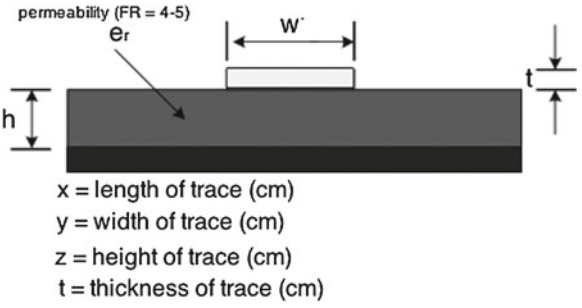


Fig. 8.6 PCB Components, interconnect two or more points: capacitance and inductance. $L(nH) \sim 2 \cdot ln\frac{5.98h}{0.8w+t}$, $C(pF) \sim \frac{0.264(e_r+1.41)}{ln(\frac{5.98h}{0.8w+t})}$ and $Z_o(\Omega) = 31.6\sqrt{\frac{L(nH)}{C(pF)}}$

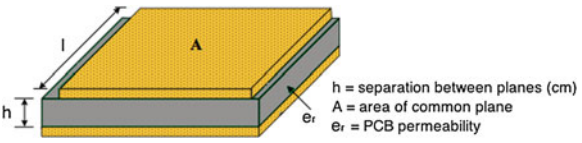


Fig. 8.7 PCB components, ground and power planes: Stray capacitance on signal traces. $C(pF) \sim \frac{0.086e_r A}{h}$

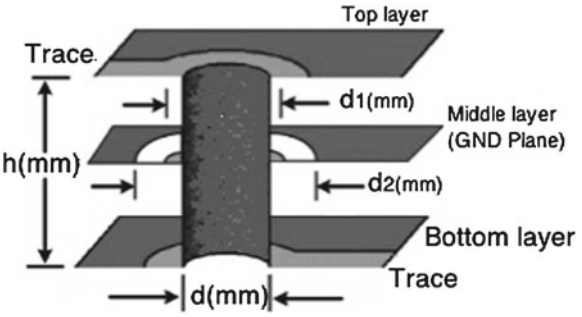


Fig. 8.8 PCB components, interconnect traces on different layers: inductance and capacitance. $L(nH) \sim \frac{h}{5} \cdot (1 + ln(\frac{4h}{d}))$ and $C(pF) \sim \frac{0.055 e_r h d_1}{d_2 - d1}$. e_r is the PCB permeability

part of PCB can be referred to passive components, for Vias that are typically very small, these elements are ignored. But, this can cause issue if the signals are in high frequency (more 100 MHz) or energy /harmonics at high frequency. The easiest way to minimize problems to via is simply not to use them with signal traces. At the very least it should be minimized (Fig. 8.8) [2–4].

The density of current flowing through a conductor is particularly important when looking at the return currents. Since there is a current flow, this will find a way return back to its source in one way or another. Return highest current density is directly under the signal trace. Even if you utilize a plan of solid mass, the concentration of current flow will still be adjacent to the signal trace.

The minimum impedance of a high-speed signal in the path is directly proportional to the trace of the PCB. This allows you to minimize the current loop area. As shown in the Fig. 8.9, the worst scenario shows a long and winding track that creates a large area of the current loop. The obvious problem is the plan of mass that is often used as a benchmark for other parts in the system. If the density of current flow is high and near to one of these reference points, then it could cause the noise in the circuit which

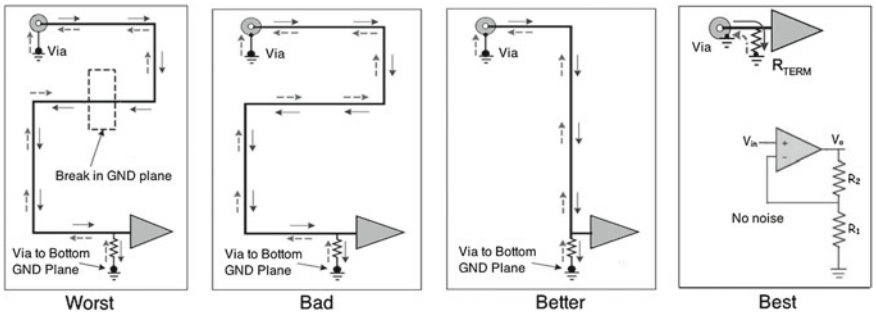


Fig. 8.9 High frequency current paths. *Continue arrow* (current flow) for top layer. *Dotted arrow* (current flow) for bottom layer

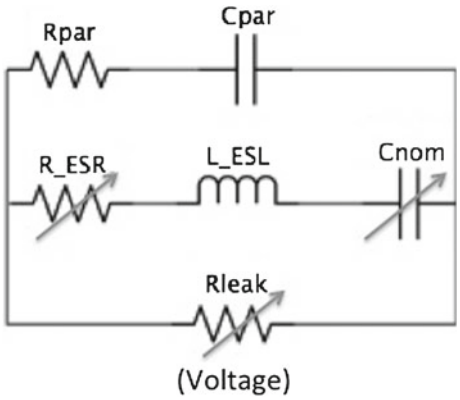


Fig. 8.10 Passive components model, capacitor

often propagates the signal. As the bad layout shown in Fig. 8.9, a long winding trace that does not follow the shortest distance between two points, it's called straight line. The better layout minimizes the distance by reducing the area of the current loop. But, the best way to do the layout is to place the receiver part as close as possible to the input. A key benefit of this method is the reference ground point for other circuits are kept 'quiet' and should have no contribution from the undesirable current flow. This also minimizes the need for adhering to strict strip-line techniques as the signal path acts as a lumped circuit and not a distributed circuit. The construction of transmission lines naturally keeps the current source and return close to each other. This helps to minimize the current loop area and dramatically reduces noise and also EMI along the path on the PCB [2–4].

PCB parasites take the form of undesired capacitors, inductors and resistors. Parasitic is extremely difficult to remove in high-speed PCB's and can destroy circuit performance. PCB prevention is the best method to minimize parasitics.

Capacitors (Fig. 8.10) are utilized extensively in the mostly systems. They are used for power-supply bypassing, AC-coupling, integrators, filtering, and so on.

The most pronounced elements are the true capacitance, the equivalent series resistance (ESR), and the equivalent series inductance (ESL). ESL is the parameter which causes the bad behavior of the capacitor at high frequencies. This is very important when ESL capacitors are used in the SMT or Surface Mount. As the lead inductance increases, the high frequency impedance limitation also increases. This increase is directly proportional to the amount of lead inductance increase.

The material of the capacitor has a significant influence on the characteristics of the capacitor. The most widely utilization of the high frequency bypass capacitor is the ceramic capacitor. These typically come in the following grades rated from the best quality to the worst quality; COG (or NPO), X7R, Z5U, and Y5V grades. The COG grades are considered to have the best characteristics as their change in capacitance with temperature is the flattest of all with the lowest dissipation factor (DF). Dissipation Factor is the measure of losses in a capacitor under an AC signal. It is the ratio of the ESR to the capacitive reactance and is measured with percent-

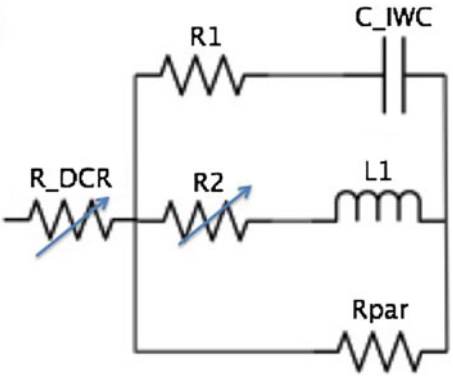


Fig. 8.11 Passive components model, inductor

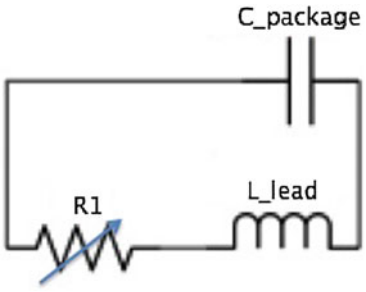


Fig. 8.12 Passive components model, resistor

age. Just as capacitors have other elements within them, inductors also have other elements. This includes the DC resistance (DCR) and the intertwining capacitance (IWC). Just as the capacitor stops behaving like a capacitor at high frequencies, an inductor stops behaving like an inductor at high frequencies. At the transition point the impedance will have a resonance causing a substantial rise in the impedance of the inductor (Fig. 8.11) [2–4].

Resistors (Fig. 8.12) are the elements which have frequency dependence characteristic. The capacitance is usually caused by the resistor package and the PCB mounting pads. The inductance is caused by the resistor leads and the PCB trace length. In general, these extra elements can be ignored if the resistance value is relatively low below 1 kΩ for example. But, they cannot be ignored if leaded resistors or wire wound resistors are utilized.

Techniques for high speed are very important, especially when you are working at frequencies above 1 MHz. Some common errors occur due to a problem of capacity. Having ground planes can be a good thing to reduce the inductance and creates a bypass capacitor, but if placed in the wrong place, it can be disastrous to the system. Another rule is to use the low value resistors. Using anything above several k-ohms is generally not recommended. Another rule is to use resistors of low value. Generally, using resistors of several k-ohm is not recommended. This is because even a small parasitic capacitance of 1 pF with a 10 kΩ resistor can cause a pole (or even worse

to zero) at 16 MHz, which is typically a high frequency. Finally, minimized trace lengths to avoid trace inductance which can also cause instability concerns if in the wrong spot [2–4].

8.2.1 Power Supply Bypassing

Bypassing is essential to high speed circuit performance. Capacitors right at power supply pins, provide low AC impedance to ground, local charge storage for fast rising/falling edges and minimization of transient currents in the ground plane (Fig. 8.13).

The rule of thumb of placing capacitors possibly closed to the IC power input pins should be adhered too. Otherwise there can be too much inductance and a resonance effect can take place along with the straight forward impedance increase due to the inductance. Typically the power and ground are on inner layers of the PCB and must be brought up to the IC level by vias. Using multiple vias is highly recommended for both power supply voltage connection and Ground connection. Additionally, the vias should run into the capacitor and then the IC. This forces the current flow into the capacitor. Placing vias directly on the capacitor mounting pads can be an effective way to minimize the routing area and still achieve the current flow routing.

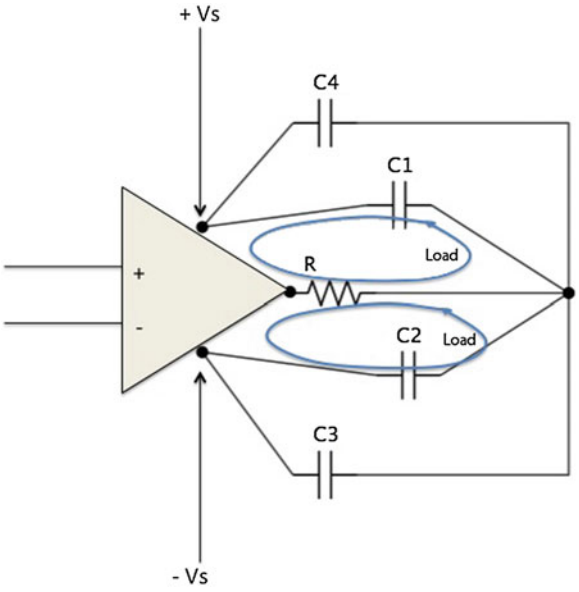


Fig. 8.13 Power supply bypassing

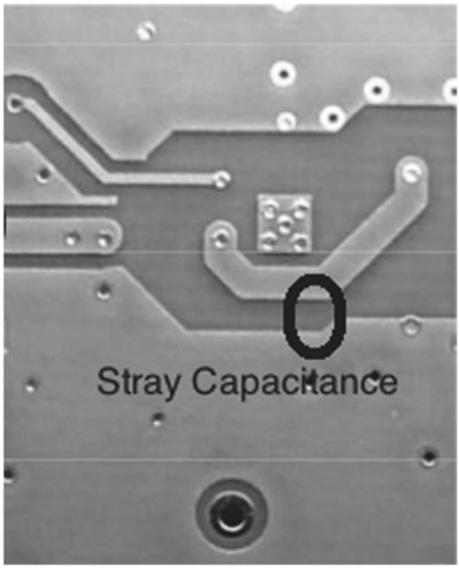


Fig. 8.14 Stray capacitance

8.2.2 Stray Capacitance

Stray capacitance is a characteristic of impedance for a transmission line. In PCB design, transmission line can be detrimental to the system and can occur noise in an amplifier which oscillation and/or decrease of amplitude of the signal [2–4]. To minimize stray capacitance, the ground plane is separated from the signal trace. This can involve in increasing the distance on the top layer, and/or removing the ground plane below the signal trace (Fig. 8.14).

8.3 Feedback Control System

A control system is about any physical system that establishes a relationship of correspondence, between a variable input (called 'reference') and an output, which is the variable control in the presence of disturbances.

The automatic control system is a set of elements interacting between them, in which the desired behavior is referred as control system and the variations that exert the control action are referred as main variables [5, 6].

A first classification of control systems is based on the kind of control:

- Closed loop control or feedback (Fig. 8.15)
- Open loop control (Fig. 8.16)

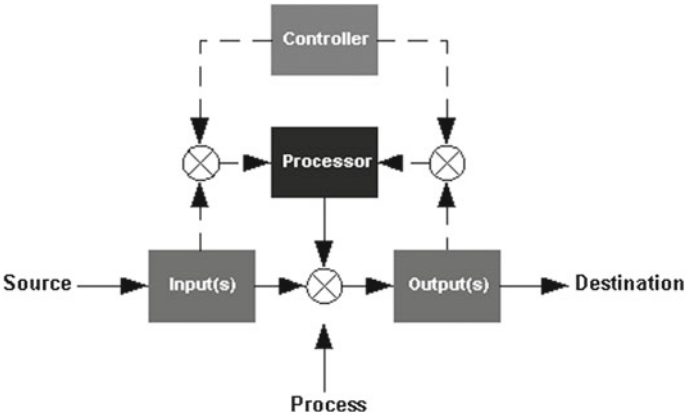


Fig. 8.15 General outline of feedback control system

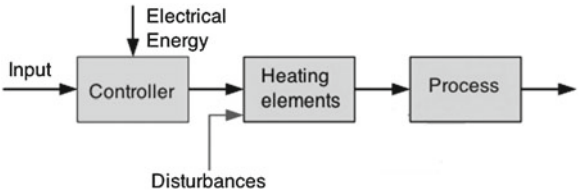


Fig. 8.16 Outline of forward control system

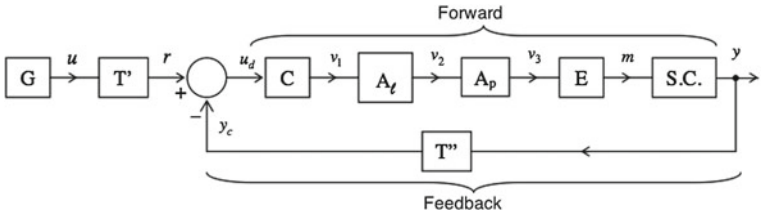


Fig. 8.17 Outline of feedback control system

The open loop control is a type of controller that computes its input into a system by using only the current state and its model of the system.

A closed-loop transfer function, instead, is a mathematical expression (algorithm) describing the net result of the effects of a closed (feedback) loop on the input signal to the circuits enclosed by the loop.

The structural diagram of a feedback control system is shown in Fig. 8.17, where the blocks have the following meaning:

- u: control variable;
- G: generator of the control variable;
- T', T": transducers, i.e. devices that modify the physical nature of the input signals that can be manipulated by electrical devices;

- r: the reference signal;
- y_c: feedback signal;
- u_d: difference signal; item C: controller or control device, which has the task of elaborating the law of control so that the magnitude of v_1 has a desired trend over time;
- A_l–A_p: amplification block to increase the amplitude of the signal to a desiderated value;
- E: actuator, which provides an output of physical quantity m to be able to drive the controlled system; can be assumed that E is a transducer power;
- S.C.: the controlled system [5, 6].

A good control system must give the value of the output variable (variable control) equal, as far as possible to that desired. It's obvious that in a real situation, this can never happen due to the noise from various internal or external sources and however due to the inherent limitations of the problem. Realistically, a good control system must respond to series of requirements or specifications that can be described as following:

Steady state error:
It is the deviation in the phase regime between the actual value of the variable control and the desired value that should be kept to a minimum value. A small error in the system coincides with a high precision. The steady state error emerges from the analysis of the response in the time domain.

Response speed:
The generic term about response speed represents a series of parameters during the transitional phase of the response (rise time, delay time, settling time) which the control system must be satisfied. The response speed is obtained from the analysis of the system in the time domain transient.

Overshoot:
It is an unexpected change of the control signal or noise and must be limited to a maximum value in order to avoid any damage or unnecessary stress. The overshoot is obtained from the analysis of the response in the time domain transient.

Stability:
It is the ability of the control system to achieve a state of equilibrium after the adjustment phase. The stability is identified by two parameters, the gain margin and phase margin which quantified how the control system is far from a state of instability. The phase margin and gain margin are obtained from the analysis in the frequency domain [5, 6].

One of the most important requirements applied to a control system is the stability, i.e. the ability of the system to reach an equilibrium state after the adjustment phase.

A linear, time-invariant with zero initial conditions system is stable if its response is limited at solicitation, otherwise, it's unstable. (BIBO stability, bounded input-bounded output).

The stability of a system can be studied in the time or frequency domain. In the first case, the stability of the system is obtained by the response in the time domain

or, more simply, the position of the poles in the complex plane s. While in the second case, it simply draws the Bode diagrams of the transfer function of the open loop system.

A linear, time invariant system with zero initial conditions is stable or asymptotically stable if its response is limited and tends to zero in correspondence of any solicitation of limited duration, otherwise it is unstable.

Usually, it is necessary to study the behavior of systems defined by open loop transfer function without poles and zeros with positive-real part. The study of stability of these systems can be performed by applying the criterion of Bode: a system in open loop with minimum phase shift, is stable if in closed loop in correspondence of ω_T, the absolute value of the transfer function's phase (ϕ_{ω_T}) is less than 180°. In particular, the system is stable if $|\phi_{\omega_T}| < 180°$, and unstable if $|\phi_{\omega_T}| > 180°$ and marginally stable if $\phi_{\omega_T}| = 180°$.

In according to the Bode criterion, therefore, it is possible to assess the stability of an open-loop system noting the Bode plots of magnitude and phase of the open loop transfer function.

The parameters used to assess the stability of a system are the phase margin and gain margin:

Phase margin:
The phase margin m_f is given by $m_f = 180° - \phi_{\omega_T}$. A system is sufficiently stable if the phase margin is greater than 30°, and unstable if the phase margin is negative.

Gain margin:
The gain margin in dB is the difference between the module in db of open loop transfer function calculated at the intersection with the abscissa (x) and the module of open loop transfer function calculated when its phase and −180°. A system is sufficiently stable if the gain margin is at least 10–20 dB. [5, 6].

References

1. Di Paolo Emilio, M. (2013). *Data acquisition system from fundamentals to applied design*. New York: Springer.
2. Razavi, B. (2008). *Fundamentals of microelectronics*. New York: Wiley.
3. Harper, A. (2000). *High performance printed circuit boards*. New York: McGrawHill.
4. Thierauf, S. C. (2004). *High-speed circuit board signal integrity*. Boston: Artech House Publishers.
5. Ellis, G. (2004). *Control system design guide*. Amsterdam: Elsevier.
6. Tewari, A. (2002). *Modern control design*. New York: Wiley.

Chapter 9
Embedded Board for High-Speed Data Acquisition and Control System

Abstract In this chapter an Embedded board Layout is presented. The goal is to realize a high speed and control system referred to the previously description.

9.1 General Layout

Data Acquisition System (Fig. 9.1) can be defined as a embedded system used for various applications: data transmission, image data acquisition and so on.

The board is managed by microcontroller while the data management is made by profile interface with on-chip A/D and D/A converters. The embedded system (Fig. 9.2) uses RISC Machines, in particular ARM processor, DSP and FPGA chips as main controlling unit in the data acquisition system.

To able archive data rates of 1,600 Mb/s, fast DDR3 memory is used to evaluate critical parameters such as: clock rise, setup/hold time, timing information related to data strobe.

The FPGAs (Figs. 9.3 and 9.4) are composed of amplifier, analog multiplexer, A/D converter and controller bus. The proposed board can be operated via ethernet bus or PCI bus to manage the data by computer.

The data acquired by A/D converter is stored in DDR3 RAM. The main task of FGPA is to generate the control signals and manage the timing signals for the whole logic system [1–3].

9.2 Hardware

The FPGAs are represented the core of the system. ADC data is sending to embedded FIFO inside the FPGA, Altera's Stratix can be used for this application.

The main function of this FPGA is buffering the input data of the ADC, facilitating the reading and writing, through a user interface. The data memorized will be sent to a microcontroller and operated according to the command.

AD9254 (ADC) can be used for this purpose. It is monolith, single 1.8 V supply, 14 bit, 150 MSPS with high performance sample and hold and on-chip voltage

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7_9

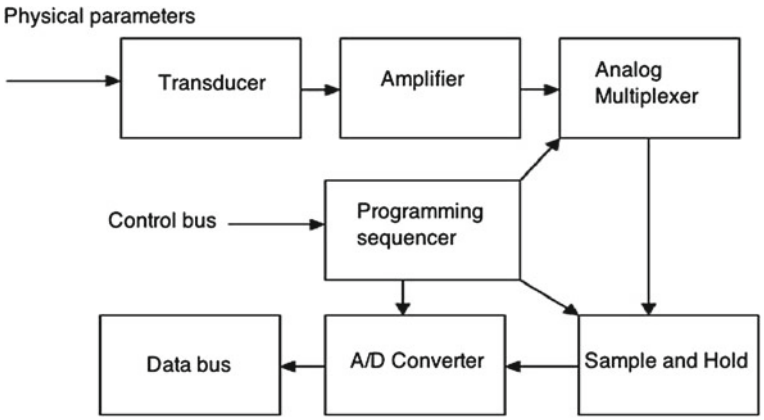


Fig. 9.1 Functional diagram of high speed data acquisition system

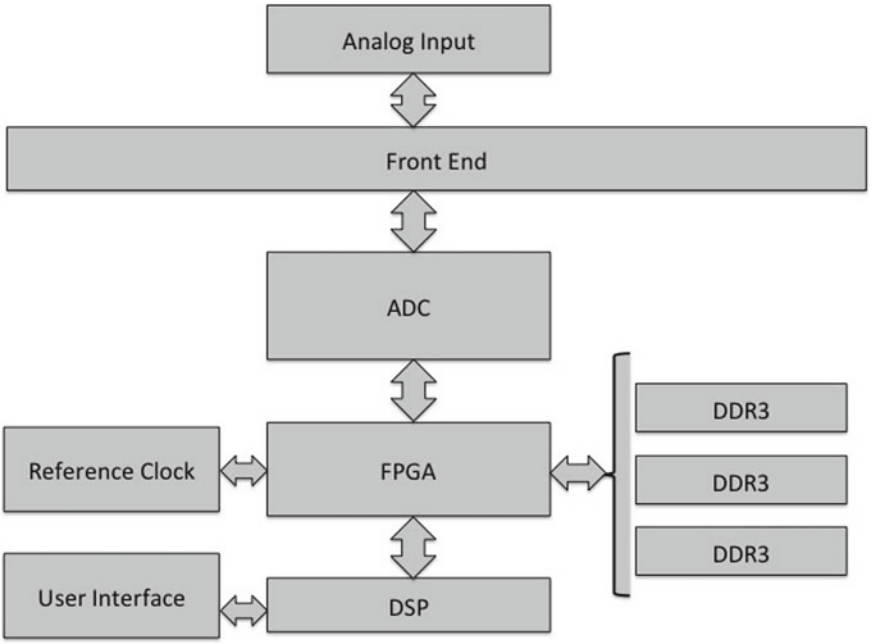


Fig. 9.2 Functional diagram of embedded part

reference. Some characteristics are the following: SNR = 71.8 dBc (72.8 dBFS) to 70 MHz input, SFDR = 84 dBc to 70 MHz input, differential input with 650 MHz bandwidth.

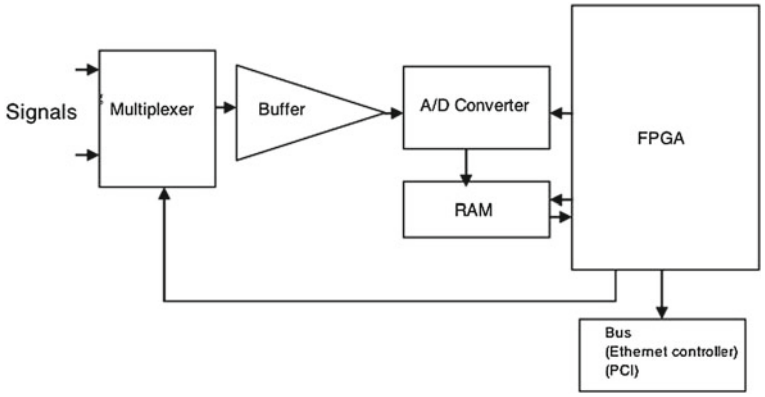


Fig. 9.3 Outline of embedded part

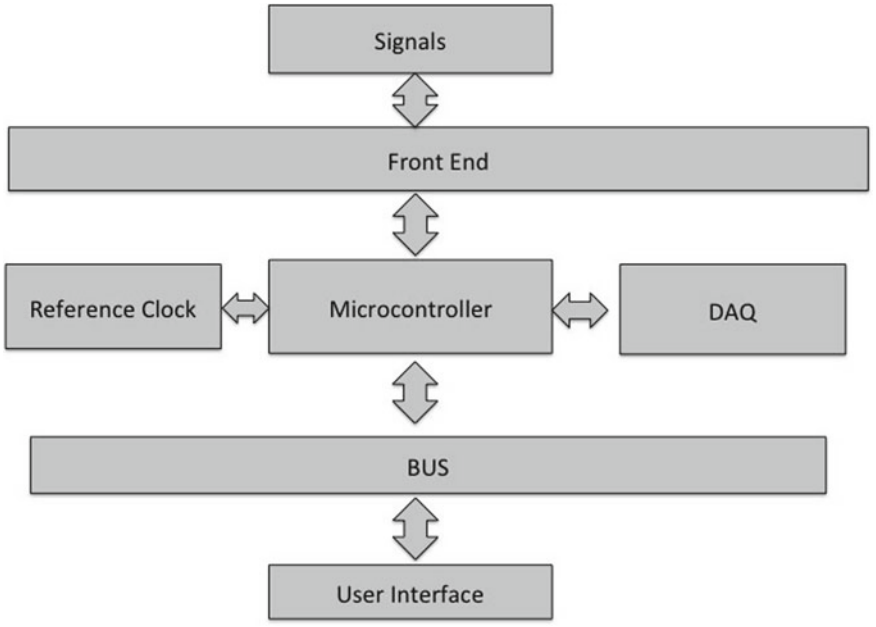


Fig. 9.4 Outline of microcontroller

Moreover, as DSP, the TMS320DM642 is used. It is highest-performance fixed-point DSP generation from Texas Instruments (TI). It is based on VelociTI very-long-instruction-word (VLIW) architecture developed by TI with 5,760 million instructions per second (MIPS) at a clock rate of 720 MHz [1–3].

9.3 Software and GUI

The process of software design consists of developing intermediate algorithms of abstraction: a set of efficiently implementable algorithms. Software package usually has analysis and presentation capabilities into the DAQ software. Your application software normally does such tasks as:

- Real-time monitoring,
- Data analysis,
- Data logging,
- Control algorithms,
- Human machine interface (HMI).

The structure of the software can be implemented for other DAQ system according to the bus interface used; if a USB interface is used it is necessary to implement C libraries (or another programming language) for USB communications.

It has two user interfaces: a text-based user interface (TUI) and a graphical user interface (GUI) designed available by means of object-oriented language (C++). Both interfaces permit DAQ management with full acquisition control. The configuration files configure the initial settings for DAQ board. Usually, it is used for user applications, server processes, and operating system settings. The GUI is a type of user interface that allows users to interact with electronic devices using images rather than text commands. The use of GUI does not require knowledge of program code and structure of configuration file. A GUI uses a combination of technologies and devices to provide a platform that the user can interact with, for the tasks of gathering and producing information. Using GUI or TUI it is possible to define DAQ setup and all functionally needs to control it.

The software interface is PCI-based. However, depending on the communication interface between FPGA and the host PC, this can be written for USB, PCIe or any other PC interface protocol [4].

9.4 Real Time Software

FreeRTOS is used in this purpose. It is a market leading real time operating system (or RTOS) from Real Time Engineers Ltd. that supports 34 architectures and receives about 1,07,000 downloads in a year. FreeRTOS is a scalable real time kernel designed specifically for small embedded systems. Its kernel is preemptive, cooperative and hybrid configuration options. An application can be designed using just tasks (Fig. 9.5), just co-routines, or a mixture of both; however tasks and co-routines use different API functions and therefore a queue (or semaphore) cannot be used to pass data from a task to a co-routine or visa versa. Co-routines (Fig. 9.6) are really only intended for use on very small processors that have severe RAM constraints.

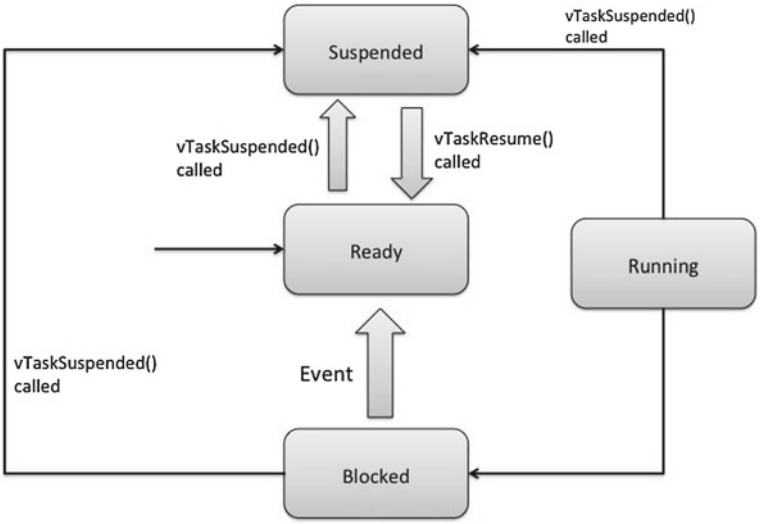


Fig. 9.5 FreeRTOS—Tasks

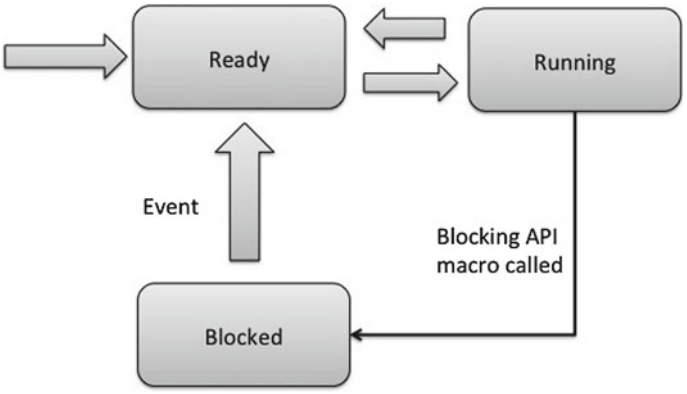


Fig. 9.6 FreeRTOS—Co-routines

A real time application that uses an RTOS can be structured as a set of independent tasks. Each task executes within its own context with no coincidental dependency on other tasks within the system or the scheduler itself.

Only one task within the application can be executing at any point in time and the real time scheduler is responsible for deciding which task this should be. The scheduler may therefore repeatedly start and stop each task (swap each task in and out) as the application executes.

Timer functionality is optional, and not part of the core FreeRTOS kernel, it is instead provided by a timer service (or daemon) task.

FreeRTOS provides a set of timer related API functions. Many of these functions use a standard FreeRTOS queue to send commands to the timer service task. The queue used for this purpose is called the 'timer command queue'. The 'timer command queue' is private to the FreeRTOS timer implementation, and cannot be accessed directly.

The FreeRTOS implementation does not execute timer callback functions from an interrupt context, unless a timer has actually expired, does not add any processing overhead to the tick interrupt.

The timer service task (primarily) makes use of existing FreeRTOS features, allowing timer functionality to be added to an application with minimal impact on the size of the application's executable binary.

9.5 Future and Improvement

Embedded system will figure prominently in the evolution of acquisition system technology makes them ideal for custom data acquisition systems and control system. In the a next step the board can be developed considering a new approach based on Android Operating System. Consumers demand for mobile computing systems and tablet computer will require special device to feature improved data acquisition capability.

Moreover, thanks to the FPGA flexibility, this board can be ideal for customized data acquisition systems and embedded applications by using a new interface and programming rules.

The small size combined with fast processing power makes this board well suited for all processes in data acquisition and offer an alternative solid solution to the traditional control systems.

Other solutions based on traditional operating systems, such as Microsoft Windows (Embedded), can be studied for easy export by the software to control the multiple control platforms.

References

1. Park, J., & Mackqy, S. (2003). *Pratical data acquisition for instrumentation and system control*. Amsterdam: Elsevier.
2. Razavi, B. (2008). *Fundamentals of microelectronics*. New York: Wiley.
3. Bin, C., et al. (1998). Architecture and design of high speed data acquisition system. *International Journal of Computer and Communication Engineering*, *1*(3), 75–84.
4. Di Paolo Emilio, M. (2013). *Data acquisition system from fundamentals to applied design*. Amsterdam: Springer.

Index

M. Di Paolo Emilio, *Embedded Systems Design for High-Speed Data Acquisition and Control*, DOI 10.1007/978-3-319-06865-7