Chapter 3

EMBEDDED SYSTEM HARDWARE

3.1 Introduction

It is one of the characteristics of embedded and cyber-physical systems that both hardware and software must be taken into account. The reuse of available hard- and software components is at the heart of the platform-based design methodology (see also page 236). Consistent with the need to consider available hardware components and with the design information flow shown in fig. 3.1, we are now going to describe some of the essentials of embedded system hardware.

![Diagram](image)

*Figure 3.1. Simplified design flow*

Hardware for embedded systems is much less standardized than hardware for personal computers. Due to the huge variety of embedded system hardware, it is impossible to provide a comprehensive overview of all types of hardware components. Nevertheless, we will try to provide a survey of some of the essential components which can be found in most systems.
In many of the cyber-physical systems, especially in control systems, hardware is used in a loop (see fig. 3.2).

![Diagram of Hardware in the Loop](image)

**Figure 3.2. Hardware in the Loop**

In this loop, information about the physical environment is made available through sensors. Typically, sensors generate continuous sequences of analog values. In this book, we will restrict ourselves to information processing where digital computers process discrete sequences of values. Appropriate conversions are performed by two kinds of circuits: sample-and-hold circuits and analog-to-digital (A/D) converters. After such conversion, information can be processed digitally. Generated results can be displayed and also be used to control the physical environment through actuators. Since most actuators are analog actuators, conversion from digital to analog signals is also needed.

This model is obviously appropriate for control applications. For other applications, it can be employed as a first order approximation. In the following, we will describe essential hardware components of cyber-physical systems following the loop structure of fig. 3.2.

### 3.2 Input

#### 3.2.1 Sensors

We start with a brief discussion of sensors. Sensors can be designed for virtually every physical quantity. There are sensors for weight, velocity, acceleration, electrical current, voltage, temperature, etc. A wide variety of physical effects can be exploited in the construction of sensors [Elsevier B.V., 2010a]. Examples include the law of induction (generation of voltages in an electric field), and photoelectric effects. There are also sensors for chemical substances [Elsevier B.V., 2010b].

Recent years have seen the design of a huge range of sensors, and much of the progress in designing smart systems can be attributed to modern sensor technology. It is impossible to cover this subset of cyber-physical hardware technology comprehensively and we can only give characteristic examples:

- **Rain sensors**: In order to remove distraction from drivers, some cars contain rain sensors. Using these, the speed of the wipers can be automatically adjusted to the amount of rain.

- **Image sensors**: There are essentially two kinds of image sensors: charge-coupled devices (CCDs) and CMOS sensors. In both cases, arrays of light sensors are used. The architecture of CMOS sensor arrays is similar to that of standard memories: individual pixels can be randomly addressed and read out. CMOS sensors use standard CMOS technology for integrated circuits [Dierickx, 2000]. Due to this, sensors and logic circuits can be integrated on the same chip. This allows some preprocessing to be done already on the sensor chip, leading to so-called smart sensors. CMOS sensors require only a single standard supply voltage and interfacing in general is easy. Therefore, CMOS-based sensors can be cheap.

In contrast, CCD technology is optimized for optical applications. In CCD technology, charges must be transferred from one pixel to the next until they can finally be read out at an array boundary. This sequential charge transfer also gave CCDs their name. For CCD sensors, interfacing is more complex.

Selecting the most appropriate image sensor is not so obvious. The image quality of CMOS sensors has been significantly improved over the recent years. Therefore, achieving a good image quality is feasible with CCD and with CMOS sensors. However, CMOS sensors are in general less power efficient than CCD sensors. Hence, if a very small power consumption is a target, CCD sensors are preferred. If minimum cost is an issue, CMOS sen-
sensors are preferred. Also, CMOS sensors are preferred if smart sensors are to be designed. Due to their smaller power consumption, compact cameras with live view displays typically use CMOS sensors [Belbachir, 2010]. For other cameras, the situation is less clear.

- **Biometric sensors:** Demands for higher security standards as well as the need to protect mobile and removable equipment have led to an increased interest in authentication. Due to the limitations of password-based security (e.g., stolen and lost passwords), smartcards, biometric sensors and biomedicine authentication receive significant attention. Biometric authentication tries to identify whether or not a certain person is actually the person they claim to be. Methods for biometric authentication include iris scans, fingerprint sensors and face recognition. Fingerprint sensors are typically fabricated using the same CMOS technology [Weste et al., 2000] which is used for manufacturing integrated circuits. Possible applications include notebooks which grant access only if the user's fingerprint is recognized [IBM, 2002]. CCD and CMOS image sensors described above are used for face recognition. False accepts as well as false rejects are an inherent problem of biometric authentication. In contrast to password-based authentication, exact matches are not possible.

- **Artificial eyes:** Artificial eye projects have received significant attention. While some projects attempt to actually affect the eye, others try to provide vision in an indirect way.

  For example, the Dobelle Institute experimented with a setup in which a little camera was attached to glasses. This camera was connected to a computer translating these patterns into electrical pulses. These pulses were then sent directly to the brain, using a direct contact through an electrode. The resolution was in the order of 128 by 128 pixels, enabling blind persons to drive a car in controlled areas [The Dobelle Institute, 2003].

  More recently, the translation of images into audio has been preferred. Obviously, it is less invasive.

- **Radio frequency identification (RFID):** RFID technology is based on the response of a tag to radio frequency signals [Hunt et al., 2007]. The tag consists of an integrated circuit and an antenna. The tag provides its identification to RFID readers. The maximum distance between tags and readers depends on the type of the tag. The technology can be applied wherever objects, animals or people should be identified.

- **Other sensors:** Other common sensors include: pressure sensors, proximity sensors, engine control sensors, Hall effect sensors, and many more.

**Embedded System Hardware**

Sensors are generating signals. Mathematically, the following definition applies:

**Definition:** A signal $\sigma$ is a mapping from the time domain $D_T$ to a value domain $D_V$:

$$\sigma : D_T \rightarrow D_V$$

Signals may be defined over a continuous or a discrete time domain as well as over a continuous or a discrete value domain.

### 3.2.2 Discretization of time:

**Sample-and-hold circuits**

All known digital computers work in a discrete time domain $D_T$. This means that they can process discrete sequences or streams of values. Hence, incoming signals over the continuous time domain must be converted to signals over the discrete time domain. This is the purpose of sample-and-hold circuits. Fig. 3.4 (left) shows a simple sample-and-hold circuit.

![Sample-and-hold-circuit](image)

*Figure 3.4. Sample-and-hold-circuit*

In essence, the circuit consists of a clocked transistor and a capacitor. The transistor operates like a switch. Each time the switch is closed by the clock signal, the capacitor is charged so that its voltage $h(t)$ is practically the same as the incoming voltage $e(t)$. After opening the switch again, this voltage will remain essentially unchanged until the switch is closed again. Each of the values stored on the capacitor can be considered as an element of a discrete sequence of values $h(t)$, generated from a continuous function $e(t)$ (see fig. 3.4, right). If we sample $e(t)$ at times $\{t_n\}$, then $h(t)$ will be defined only at those times.

An ideal sample-and-hold circuit would be able to change the voltage at the capacitor in an arbitrarily short amount of time. This way, the input voltage at a particular instance in time could be transferred to the capacitor and each element in the discrete sequence would correspond to the input voltage at a particular point in time. In practice, however, the transistor has to be kept closed for a
short time window in order to really charge or discharge the capacitor. The
voltage stored on the capacitor will then correspond to a voltage reflecting that
short time window.

An interesting question is this one: would we be able to reconstruct the original
signal \( e(t) \) from the sampled signal \( h(t) \)? At this time, we revert to the fact that
arbitrary signals can be approximated by summing (possibly phase-shifted)
sine functions of different frequencies (Fourier approximation)\(^1\). For example,
fig. 3.5 and fig. 3.6 demonstrate how even a square wave can be approximated
by sine waves of increasing frequencies.

![Figure 3.5](image)

Figure 3.5. Approximation of a square wave by sine waves for \( K=1 \) (left) and \( K=3 \) (right)

![Figure 3.6](image)

Figure 3.6. Approximation of a square wave by sine waves for \( K=7 \) (left) and \( K=11 \) (right)

These graphs display a graphical representation of equation 3.1 [Oppenheim
et al., 2009], where \( p \) is the period:

\[
e^\prime_K(t) = \sum_{k=1,3,5,7,9,...}^{K} \left( \frac{4}{\pi k} \sin \left( \frac{2\pi k t}{p} \right) \right)
\]  

(3.1)

\(^1\)The presentation in this book is based on the assumption that a full presentation of the theory of Fourier
approximations cannot be included in a course on embedded systems. Therefore, only the impact of this
theory is demonstrated by using examples. Students would benefit from knowing the theory behind these
examples.

Embedded System Hardware

A data processing transformation \( Tr \) is said to be linear, if for signals \( e_1(t) \)
and \( e_2(t) \) we have:

\[
Tr(e_1 + e_2) = Tr(e_1) + Tr(e_2)
\]  

(3.2)

In the following, we restrict ourselves to linear systems. Then, in order to
answer the question raised above, we study the effect of sampling on each of
the sine waves independently.

Suppose that our input signal corresponds to either of the two functions \( e_3 \) or
\( e_4 \):

\[
e_3(t) = \sin \left( \frac{2\pi t}{8} \right) + 0.5 \sin \left( \frac{2\pi t}{4} \right)
\]

(3.3)

\[
e_4(t) = \sin \left( \frac{2\pi t}{8} \right) + 0.5 \sin \left( \frac{2\pi t}{4} \right) + 0.5 \sin \left( \frac{2\pi t}{1} \right)
\]

(3.4)

The sine waves used in these functions have periods of \( p = 8, 4, \) and \( 1 \), respectively (this can be seen easily by comparing these sine waves with those used
in eq. 3.1). A graphical representation of these functions is shown in fig. 3.7.

![Figure 3.7](image)

Figure 3.7. Visualization of functions \( e_3(t) \) (solid) and \( e_4(t) \) (dotted)

Suppose that we will be sampling these signals at integer times. It then so
happens that both signals have the same value whenever they are sampled.
Obviously, it is not possible to distinguish between \( e_3(t) \) and \( e_4(t) \) if we sample
at these instances in time as shown and if only the sampled signal is available.
In general, sampled signals will not allow us to distinguish between some slow signal \( e_3(t) \) and some other faster varying signal \( e_4(t) \) if \( e_3(t) \) and \( e_4(t) \) are identical each time we are sampling the signals. The fact that two or more unsampled signals can have the same sampled representation is called **aliasing**. We are not sampling \( e_4(t) \) frequently enough to notice, for example, that it has slope changes between integer times. So, from this counterexample we can conclude that reconstruction of the original unsampled signal is not feasible unless we have additional knowledge about the frequencies or the waveforms present in the input signal.

How frequently do we have to sample signals to be able to distinguish between different sine waves?

Let us assume that we are sampling the input signal at constant time intervals, such that \( p_s \) is the sampling period\(^2\):

\[
\forall s : \quad p_s = t_{s+1} - t_s
\]  

(3.5)

Let

\[
f_s = \frac{1}{p_s}
\]

(3.6)

be the sampling rate or sampling frequency.

According to the theory of sampling [Oppenheim et al., 2009], aliasing is avoided if we restrict the frequencies of the incoming signal to less than half of the sampling frequency \( f_s \):

\[
p_s < \frac{p_N}{2} \quad \text{where } p_N \text{ is the period of the "fastest" sine wave, or}
\]

(3.7)

\[
f_s > 2f_N \quad \text{where } f_N \text{ is the frequency of the "fastest" sine wave}
\]

(3.8)

**Definition:** \( f_N \) is called the Nyquist frequency, \( f_s \) is the sampling rate.

The condition in equation 3.8 is called **sampling criterion**, and sometimes the **Nyquist sampling criterion**.

Therefore, reconstruction of input signals \( e(t) \) from discrete samples \( h(t) \) can be successful only if we make sure that higher frequency components such as the one in \( e_4(t) \) are removed. This is the purpose of anti-aliasing filters. Anti-aliasing filters are placed in front of the sample-and-hold circuit (see fig. 3.8).

![Figure 3.8. Anti-aliasing placed in front of the sample-and-hold circuit](image)

Fig. 3.9 demonstrates the ratio between the amplitudes of the output and the input waves as a function of the frequency for this filter.

![Figure 3.9. Ideal and realizable anti-aliasing filters (low-pass filters)](image)

Ideally, such a filter would remove all frequencies at and above the sampling frequency and keep all other components unchanged. This way, it would convert signal \( e_4(t) \) into signal \( e_3(t) \). In practice, such ideal filters do not exist. Realizable filters will already start attenuating frequencies smaller than \( f_s/2 \) and will still not eliminate all frequencies larger than \( f_s/2 \) (see fig. 3.9). Attenuated high frequency components will exist even after filtering. For frequencies smaller than \( f_s/2 \), there may also be some “overshoot”, i.e. frequencies for which there is some amplification of the input signal. The design of good anti-aliasing filters is an art by itself.

### 3.2.3 Discretization of values: A/D-converters

Since we are restricting ourselves to digital computers, we must also replace signals that map time to a continuous value domain \( D_Y \) by signals that map time to a discrete value domain \( D_Y' \). This conversion from analog to digital values is done by analog-to-digital (A/D) converters. There is a large range of A/D converters with varying speed/precision characteristics. In this book, we will present two extreme cases:

---

\(^2\)In order to be consistent with the notation in scheduling theory, we denote the period by \( p_s \) instead of by \( T_s \). The latter notation is frequently used in digital signal processing.
Flash A/D converter: This type of A/D converters uses a large number of comparators. Each comparator has two inputs, denoted as + and -. If the voltage at input + exceeds that at input -, the output corresponds to a logical '1' and it corresponds to a logical '0' otherwise.

In the A/D-converter, all inputs are connected to a voltage divider. If input voltage \( h(t) \) exceeds \( \frac{3}{4} V_{ref} \), the comparator at the top of fig. 3.10 (a) will generate a '1'. The encoder at the output of the comparators will try to identify the most significant '1' and will encode this case as the largest output value. The case \( h(t) > V_{ref} \) should normally be avoided since \( V_{ref} \) is typically close to the supply voltage of the circuit and input voltages exceeding the supply voltage can lead to electrical problems. In our case, input voltages larger than \( V_{ref} \) generate the largest digital value as long as the converter does not fail due to the high input voltage.

![Figure 3.10. (a) Flash A/D converter (b) w as a function of h](image)

Now, if input voltage \( h(t) \) is less than \( \frac{3}{4} V_{ref} \), but still larger than \( \frac{1}{4} V_{ref} \), the comparator at the top of fig. 3.10 will generate a '0', while the next comparator will still signal a '1'. The encoder will encode this as the second-largest value.

Similar arguments hold for cases \( \frac{1}{4} V_{ref} < h(t) < \frac{3}{4} V_{ref} \) and \( 0 < h(t) < \frac{1}{4} V_{ref} \), which will be encoded as the third-largest and the smallest value, respectively. Fig. 3.10 (b) shows the relation between input voltages and generated digital values.

The outputs of the comparators encode numbers in a special way: if a certain comparator output is equal to '1', then all the less significant outputs are all equal to '1'. The encoder transforms this representation of numbers into the usual representation of natural numbers. The encoder is actually a so-called "priority encoder", encoding the most significant input number carrying a '1' in binary.

The circuit can convert positive analog input voltages into digital values. Converting both positive and negative voltages and generating two's complement numbers requires some extensions.

A/D-converters are characterized by their resolution. This term has several different but related meanings [Analog Devices Inc. Eng., 2004]. The resolution (measured in bits) is the number of bits produced by an A/D-converter. For example, A/D-converters with a resolution of 16 bits are needed for many audio applications. However, the resolution is also measured in volts, and in this case it denotes the difference between two input voltages causing the output to be incremented by 1:

\[
Q = \frac{V_{FSR}}{n}
\]

Where:

- \( V_{FSR} \) : is the difference between the largest and the smallest voltage.
- \( Q \) : is the resolution in volts per step, and
- \( n \) : is the number of voltage intervals (not the number of bits).

Example: For the A/D-converter of fig. 3.10, the resolution is 2 bits or \( \frac{1}{4} V_{ref} \) volts, if we assume \( V_{ref} \) as the largest voltage.

The key advantage of the flash A/D-converter is its speed. It does not need any clock. The delay between the input and the output is very small and the circuit can be used easily, for example, for high-speed video applications. The disadvantage is its hardware complexity: we need \( n - 1 \) comparators in order to distinguish between \( n \) values. Imagine using this circuit in generating digital audio signals for CD recorders. We would need \( 2^{16} - 1 \) comparators! High-resolution A/D-converters must be built in a different way.

**Successive approximation:** Distinguishing between a large number of digital values is possible with A/D converters using successive approximation. The circuit is shown in fig. 3.11.

---

1 In practice, the case of equal voltages is not relevant, as the actual behavior for very small differences between the voltages at the two inputs depends on many factors (like temperatures, manufacturing processes etc.) anyway.

2 Such encoders are also useful for finding the most significant '1' in the mantissa of floating point numbers.
Figure 3.11. Circuit using successive approximation.

The key idea of this circuit is to use binary search. Initially, the most significant output bit of the successive approximation register is set to '1', all other bits are set to '0'. This digital value is then converted to an analog value, corresponding to 0.5× the maximum input voltage. If h(t) exceeds the generated analog value, the most significant bit is kept at '1', otherwise it is reset to '0'.

This process is repeated with the next bit. It will remain set to '1' if the input value is either within the second or the fourth quarter of the input value range. The same procedure is repeated for all the other bits. Fig. 3.12 shows an example.

In Fig. 3.12, initially the most significant bit is set to '1'. This value is kept, since the resulting V₀ < h(t). Then, the second-most significant bit is set to '1'. It is reset to '0', since the resulting V₀ ≥ h(t). Next, the third-most significant bit is tried, and so on. Obviously, h(t) must be constant during the conversion. This requirement is met if we employ a sample-and-hold circuit as shown above. The resulting digital signal is called w(t).

The key advantage of the successive approximation technique is its hardware efficiency. In order to distinguish between n digital values, we need \( \log_2(n) \) bits in the successive approximation register and the D/A converter. The disadvantage is its speed, since it needs \( O(\log_2(n)) \) steps. These converters can therefore be used for high-resolution applications, where moderate speeds are required. Examples include audio applications.

Fig. 3.13 highlights the behavior of a D/A-converter when the input signal is that of equation 3.3. Only the behavior for a positive input signal is shown.

The figure includes the voltage corresponding to the digital value, the original voltage, and the difference between the two. Obviously, the converter is "truncating" the digital representation of the analog signal to the number of available bits (i.e., the digital value is always less than or equal to the analog value). This is a consequence of the way in which we are doing comparisons. "Rounding" converters would need an internal correction by "half a bit".

Effectively, the digital signal encodes values corresponding to the sum of the original analog values and the difference \( w(t) - h(t) \). This means, it appears as if the difference between the two signals had been added to the original signal. This difference is a signal called quantization noise:

\[
\text{quantization noise}(t) = w(t) - h(t)
\] (3.9)
\[(\text{quantization noise}(t)) < Q\] (3.10)

Obviously, it is possible to decrease quantization noise by increasing the resolution (in bits) of the A/D-converters. The impact of quantization noise is frequently captured in the definition of the signal-to-noise ratio (SNR). The SNR is measured in decibels (tens of a Bel, named after Alexander G. Bell):

\[
\text{SNR (in decibels)} = 10 \cdot \log \left( \frac{\text{power of the "useful" signal}}{\text{power of the noise signal}} \right) \quad (3.11)
\]

\[
= 20 \cdot \log \left( \frac{\text{voltage of the "useful" signal}}{\text{voltage of the noise signal}} \right) \quad (3.12)
\]

In this case, we have used the fact that, for any given impedance \(R\), the power of a signal is equal to the square of the voltage. Decibels are no physical units, since the signal-to-noise ratio is dimension-less.

For any signal \(h(t)\), the power of the quantization noise is equal to \(\alpha \cdot Q\), where \(\alpha \leq 1\) depends on the waveform of \(h(t)\). If \(h(t)\) can always be represented exactly by a digital value, \(\alpha = 0\). If \(h(t)\) is always "just a little" below the next value that can be represented, \(\alpha\) may be close to 1.

For example (for \(\alpha \sim 1\)), the SNR of 16-bit CD audio is in the order of:

\[20 \cdot \log(2^{16}) = 96 \text{ decibels (dB)}\]

For high-quality 24-bit CDs we would obtain an SNR of about 144 dB. Values of \(\alpha < 1\) and imperfections of A/D-converters may change these numbers a bit.

There are several other types of A/D-converters. They differ by their speed and their precision [O’Neill, 2006]. Techniques for automatically selecting the most appropriate converter exist [Vogels and Gielen, 2003].

3.3 Processing Units

3.3.1 Overview

Currently available embedded systems require electrical energy to operate. The amount of electrical energy used is frequently called “consumed energy”. Strictly speaking, this term is not correct, since this electrical energy is converted to other forms of energy, typically thermal energy. For embedded systems, energy availability is a deciding factor. This was already observed in a Dutch road mapping effort: “Power is considered as the most important constraint in embedded systems” [Eggermont, 2002]. The importance of power and energy efficiency was initially recognized for embedded systems. The focus on these objectives was later taken up for general purpose computing as well and led to initiatives such as the green computing initiative. For information processing in embedded systems, we will consider ASICs (application-specific integrated circuits) using hardwired multiplexed designs, reconfigurable logic, and programmable processors. These three technologies are quite different as far as their energy efficiency is concerned. Fig. 3.14 repeats the information already provided on page 6.

Fig. 3.14 reflects the efficiency/flexibility conflict of currently available hardware technologies: if we want to aim at very power- and energy-efficient designs, we should use ASICs instead of flexible designs based on processors or re-programmable logic. If we go for excellent flexibility, we cannot be power-efficient.

The energy \(E\) for a certain application is closely related to the power \(P\) required per operation, since

\[E = \int P dt\] (3.13)
Let us assume that we start with some design having a power consumption of \( P_0(t) \), leading to an energy consumption of

\[
E_0 = \int_0^{t_0} P_0(t) \, dt
\]

after \( t_0 \) units of execution time. Suppose that a modified design finishing computations already at time \( t_1 \) comes with a power consumption of \( P_1(t) \) and an energy consumption of

\[
E_1 = \int_0^{t_1} P_1(t) \, dt
\]

If \( P_1(t) \) is not too much larger than \( P_0(t) \), then a reduction of the execution time also reduces the energy consumption. However, in general this is not necessarily always true. The situation is also shown in fig. 3.15: \( E_1 \) may be smaller than \( E_0 \), but \( E_1 \) can also be larger than \( E_0 \).

![Figure 3.15. Comparison of energies \( E_0 \) and \( E_1 \)](image)

Minimization of power and energy consumption are both important. Power consumption has an effect on the size of the power supply, the design of the voltage regulators, the dimensioning of the interconnect, and short term cooling. Minimizing the energy consumption is required especially for mobile applications, since battery technology is only slowly improving [ITRS Organization, 2009], and since the cost of energy may be quite high. Also, a reduced energy consumption decreases cooling requirements and improves the reliability (since the lifetime of electronic circuits decreases for high temperatures). We will consider ASICs first.

### 3.3.2 Application-Specific Circuits (ASICs)

For high-performance applications and for large markets, application-specific integrated circuits (ASICs) can be designed. However, the cost of designing and manufacturing such chips is quite high. For example, the cost of the mask set which is used for transferring geometrical patterns onto the chip can amount to about \( 10^5 \cdot 10^6 \) Euros or dollars. In fact, the cost for mask sets has grown exponentially over the recent years. Also, this approach suffers from long design times and the lack of flexibility: correcting design errors typically requires a new mask set and a new fabrication run. Therefore, ASICs are appropriate only if either maximum energy efficiency is needed and if the market accepts the costs or if a large number of such systems can be sold. Consequently, the design of ASICs is not covered in this book.

### 3.3.3 Processors

The key advantage of processors is their flexibility. With processors, the overall behavior of embedded systems can be changed by just changing the software running on those processors. Changes of the behavior may be required in order to correct design errors, to update the system to a new or changed standard or in order to add features to the previous system. Because of this, processors have become very popular.

Embedded processors must be efficient and they do not need to be instruction set compatible with commonly used personal computers (PCs). Therefore, their architectures may be different from those processors found in PCs. Efficiency has a number of different aspects (see page 5):

- **Energy-efficiency**: Architectures must be optimized for their energy-efficiency and we must make sure that we are not losing efficiency in the software generation process. For example, compilers generating 50% overhead in terms of the number of cycles will take us further away from the efficiency of ASICs, possibly by even more than 50%, if the supply voltage and the clock frequency must be increased in order to meet timing deadlines.

  There is a large amount of techniques available that can make processors energy efficient and energy efficiency should be considered at various levels of abstraction, from the design of the instruction set down to the design of the chip manufacturing process [Burd and Brodersen, 2003]. Gated clocking is an example of such a technique. With gated clocking, parts of the processor are disconnected from the clock during idle periods. For example, no clock is applied to direct memory access (DMA) hardware or bus bridges if they are not needed. Also, there are attempts, to get rid of the clock for ma-
EMBEDDED SYSTEM DESIGN

CMOS circuits can be approximated as [Chandrakasan et al., 1992], [Chandrakasan et al., 1995]:
\[ \tau = k \cdot C_L \cdot \frac{V_{dd}}{(V_{dd} - V_I)^2} \] (3.15)

where \( k \) is a constant, and \( V_I \) is the threshold voltage. \( V_I \) has an impact on the transistor input voltage required to switch the transistor on. For example, for a maximum supply voltage of \( V_{dd,max} = 3.3 \text{ volts} \), \( V_I \) may be in the order of 0.8 volts. Consequently, the maximum clock frequency is a function of the supply voltage. However, decreasing the supply voltage reduces the power quadratically, while the run-time of algorithms is only linearly increased (ignoring the effects of the memory system). This can be exploited in a technique called dynamic voltage scaling (DVS). For example, the Crusoe processor by Transmeta [Klaiber, 2000] provided 32 voltage levels between 1.1 and 1.6 volts, and the clock could be varied between 200 MHz and 700 MHz in increments of 33 MHz. Transitions from one voltage/frequency pair to the next took about 20 ms. Design issues for DVS-capable processors are described in a paper by Burd and Brodersen [Burd and Brodersen, 2000]. According to the same paper, potential power savings will exist even for future technologies with a decreased maximum \( V_{dd} \), since the threshold voltages will also be decreased (unfortunately, this will lead to increased leakage currents, increasing the standby power consumption). In 2004, six different speed/voltage pairs were provided with the Intel SpeedStep\textsuperscript{T\textregistered} technology for the Pentium\textsuperscript{M} [Intel, 2004].

- **Code-size efficiency:** Minimizing the code size is very important for embedded systems, since hard disk drives are typically not available and since the capacity of memory is typically also very limited.\(^\text{6}\) This is even more pronounced for systems on a chip (SoCs). For SoCs, the memory and processors are implemented on the same chip. In this particular case, memory is called embedded memory. Embedded memory may be more expensive to fabricate than separate memory chips, since the fabrication processes for memories and processors must be compatible. Nevertheless, a large percentage of the total chip area may be consumed by the memory. There are several techniques for improving the code-size efficiency:

- **CISC machines:** Standard RISC processors have been designed for speed, not for code-size efficiency. Earlier Complex Instruction Set

\(^\text{6}\)The availability of large flash memories makes memory size constraints less tight.
Processors (CISC machines) were actually designed for code-size efficiency, since they had to be connected to slow memories. Caches were not frequently used. Therefore, "old-fashioned" CISC processors are finding applications in embedded systems. ColdFire processors [Freescale semiconductor, 2005], which are based on the Motorola 68000 family of CISC processors are an example.

- **Compression techniques**: In order to reduce the amount of silicon needed for storing instructions as well as in order to reduce the energy needed for fetching these instructions, instructions are frequently stored in the memory in compressed form. This reduces both the area as well as the energy necessary for fetching instructions. Due to the reduced bandwidth requirements, fetching can also be faster. A (hopefuly small and fast) decoder is placed between the processor and the (instruction) memory in order to generate the original instructions on the fly (see fig. 3.17, right). Instead of using a potentially large memory of uncompressed instructions, we are storing the instructions in a compressed format.

![Figure 3.17. Decompression of compressed instructions](image)

The goals of compression can be summarized as follows:

- We would like to save ROM and RAM areas, since these may be more expensive than the processors themselves.
- We would like to use some encoding technique for instructions and possibly also for data with the following properties:
  - There should be little or no run-time penalty for these techniques.

---

7 We continue denoting multiplexers, arithmetic units and memories by shape symbols, due to their widespread use in technical documentation. For memories, we adopt shape symbols including an explicit address decoder (included in the shape symbols for the ROMs on the right). These decoders identify the address input.

Embedded System Hardware

- Decoding should work from a limited context (it is, for example, impossible to read the entire program to find the destination of a branch instruction).
- Word-sizes of the memory, of instructions and addresses must be taken into account.
- Branch instructions branching to arbitrary destination addresses must be supported.
- Fast encoding is only required if writable data is encoded. Otherwise, fast decoding is sufficient.

There are several variations of this scheme:

* For some processors, there is a **second instruction set**. This second instruction set has a narrower instruction format. An example of this is the ARM processor family. The ARM instruction set is a 32 bit instruction set. The ARM instruction set includes predicated execution. This means an instruction is executed if and only if a certain condition is met (see page 148). This condition is encoded in the first four bits of the instruction format. Most ARM processors also provide a second instruction set, with 16 bit wide instructions, called THUMB instructions. THUMB instructions are shorter, since they do not support predication, use shorter and less register fields and use shorter immediate fields (see fig. 3.18).

![Figure 3.18. Re-encoding THUMB into ARM instructions](image)

THUMB instructions are dynamically converted into ARM instructions while programs are decoded. THUMB instructions can use only half the registers in arithmetic instructions. Therefore, register fields of THUMB instructions are concatenated with a "0"-bit. In the THUMB instruction set, source and destination registers are identical and the length of constants that can be used, is reduced by 4 bits. During decoding, pipelining is used to keep the run-time penalty low.

8 Using VHDL-notation (see page 80), concatenation is denoted by an &-sign and constants are enclosed in quotes in fig. 3.18.
Similar techniques also exist for other processors. The disadvantage of this approach is that the tools (compilers, assemblers, debuggers etc.) must be extended to support a second instruction set. Therefore, this approach can be quite expensive in terms of software development cost.

A second approach is the use of **dictionaries**. With this approach, each instruction pattern is stored only once. For each value of the program counter, a look-up table then provides a pointer to the corresponding instruction in the instruction table, the dictionary (see fig. 3.19).

![Diagram of dictionary approach for instruction compression](image)

**Figure 3.19. Dictionary approach for instruction compression**

This approach relies on the idea that only very few different instruction patterns are used. Therefore, only few entries are required for the instruction table. Correspondingly, the bit width of the pointers can be quite small. Many variations of this scheme exist. Some are called **two-level control store** [Dasgupta, 1979], **nanoprogramming** [Stritter and Gunter, 1979], or **procedure ex-lining** [Vahid, 1995].


- **Run-time efficiency**: In order to meet time constraints without having to use high clock frequencies, architectures can be customized to certain application domains, such as digital signal processing (DSP). One can even go one step further and design application-specific instruction set processors (ASIPs). As an example of domain-specific processors, we will consider processors for DSP. In digital signal processing, digital filtering is a very frequent operation. Let us assume that we are extending the processing pipeline of fig. 3.4 by such filtering. Naming conventions for the involved signals are shown in fig. 3.20.

Equation 3.16 describes a digital filter generating an output signal $x(t)$ from an input signal $w(t)$. Both signals are defined over the (usually unbounded)

\[
x(t) = \sum_{k=0}^{n-1} w(t-k) \cdot a_k
\]

(3.16)

A certain output element $x(t)$ corresponds to a weighted average over the last $n$ signal elements of $w$ and can be computed iteratively, adding one product at a time. Processors for DSP are designed such that each iteration can be encoded as a single instruction. Let us consider an example. Fig. 3.21 shows the internal architecture of an ADSP 2100 DSP processor.

![Diagram of internal architecture of the ADSP 2100 processor](image)

**Figure 3.21. Internal architecture of the ADSP 2100 processor**

The processor has two memories, called D and P. A special address generating unit (AGU) can be used to provide the pointers for accessing these memories. There are separate units for additions and multiplications, each
generated which cannot be returned in any of the standard registers. The result register will contain a pattern of all zeros. No result could be further away from the true result than this one.

```
+     0111
1001
Standard wrap-around arithmetic
saturating arithmetic
10000
1111
```

*Figure 3.23. Wrap-around vs. saturating arithmetic for unsigned integers*

In saturating arithmetic, we try to return a result which is as close as possible to the true result. For saturating arithmetic, the largest value is returned in the case of an overflow and the smallest value is returned in the case of an underflow. This approach makes sense especially for video and audio applications: the user will hardly recognize the difference between the true result value and the largest value that can be represented. Also, it would be useless to raise exceptions if overflows occur, since it is difficult to handle exceptions in real-time. Note that we need to know whether we are dealing with signed or unsigned add instructions in order to return the right value.

- **Fixed-point arithmetic:** Floating-point hardware increases the cost and power-consumption of processors. Consequently, it has been estimated that 80% of the DSP processors do not include floating-point hardware [Aamodt and Chow, 2000]. However, in addition to supporting integers, many such processors do support fixed-point numbers. Fixed-point data types can be specified by a 3-tuple \((wf, iwf, sign)\), where \(wf\) is the total word-length, \(iwf\) is the integer word-length (the number of bits left of the binary point), and sign \(s \in \{s, u\}\) denotes whether we are dealing with unsigned or signed numbers. See also fig. 3.24. Furthermore, there may be different rounding modes (e.g. truncation) and overflow modes (e.g. saturating and wrap-around arithmetic).

```
\(s\) \hspace{1cm} \text{binary point}
\(iwf\) \hspace{1cm} \(fwf\)
```

*Figure 3.24. Parameters of a fixed-point number system*

For fixed-point numbers, the position of the binary point is maintained after multiplications (some low order bits are truncated or rounded). For fixed-point processors, this operation is supported by hardware.

- **Real-time capability:** Some of the features of modern processors used in PCs are designed to improve the average execution time of programs. In many cases, it is difficult if not impossible to formally verify that they improve the worst case execution time. In such cases, it may be better not to implement these features. For example, it is difficult (though not impossible [Abramson, 2002]) to guarantee a certain speedup resulting from the use of caches. Therefore, many embedded processors do not have caches. Also, virtual addressing and demand paging are normally not found in embedded systems. Techniques for computing worst case execution times will be presented in section 5.2.2.

- **Multiple memory banks or memories:** The usefulness of multiple memory banks was demonstrated in the ADSP 2100 example: the two memories D and P allow fetching both arguments at the same time. Several DSP processors come with two memory banks.

- **Heterogeneous register files:** Heterogeneous register files were already mentioned for the filter application.

- **Multiply/accumulate instructions:** These instructions perform multiplications followed by additions. They were also already used in the filter application.

### 3.3.3.2 Multimedia processors/instruction sets

Registers and arithmetic units of many modern architectures are at least 64 bits wide. Therefore, two 32 bit data types ("double words"), four 16 bit data types ("words") or eight 8 bit data types ("bytes") can be packed into a single register (see fig. 3.25).

```
\begin{array}{cccc}
\text{word 3} & \text{word 2} & \text{word 1} & \text{word 0} \\
\end{array}
```

*Figure 3.25. Using 64 bit registers for packed words*

Arithmetic units can be designed such that they suppress carry bits at double word, word or byte boundaries. Multimedia instruction sets exploit this fact by supporting operations on packed data types. Such instructions are sometimes called single-instruction, multiple-data (SIMD) instructions, since a single instruction encodes operations on several data elements. With bytes packed into
64-bit registers, speed-ups of up to about eight over non-packed data types are possible. Data types are typically stored in packed form in memory. Unpacking and packing are avoided if arithmetic operations on packed data types are used. Furthermore, multimedia instructions can usually be combined with saturating arithmetic and therefore provide a more efficient form of overflow handling than standard instructions. Hence, the overall speed-up achieved with multimedia instructions can be significantly larger than the factor of eight enabled by operations on packed data types. Due to the advantages of operations on packed data types, new instructions have been added to several processors. For example, so-called streaming SIMD extensions (SSE) have been added to Intel's family of Pentium®-compatible processors [Intel, 2008]. New instructions have also been called short vector instructions. Currently (in 2010), Intel® Advanced Vector Extensions (AVX) are being introduced [Intel, 2010a].

3.3.3.3 Very long instruction word (VLIW) processors

Computational demands for embedded systems are increasing, especially when multimedia applications, advanced coding techniques or cryptography are involved. Performance improvement techniques used in high-performance microprocessors are not appropriate for embedded systems: driven by the need for instruction set compatibility, processors found, for example, in PCs spend a huge amount of resources and energy on automatically finding parallelism in application programs. Still, their performance is frequently not sufficient. For embedded systems, we can exploit the fact that instruction set compatibility with PCs is not required. Therefore, we can use instructions which explicitly identify operations to be performed in parallel. This is possible with explicit parallelism instruction set computers (EPICS). With EPICS, detection of parallelism is moved from the processor to the compiler. This avoids spending silicon and energy on the detection of parallelism at runtime. As a special case, we consider very long instruction word (VLIW) processors. For VLIW processors, several operations or instructions are encoded in a long instruction word (sometimes called instruction packet) and are assumed to be executed in parallel. Each operation/instruction is encoded in a separate field of the instruction packet. Each field controls certain hardware units. Four such fields are used in fig. 3.26, each one controlling one of the hardware units.

For VLIW architectures, the compiler has to generate instruction packets. This requires that the compiler is aware of the available hardware units and to schedule their use.

Instruction fields must be present, regardless of whether or not the corresponding functional unit is actually used in a certain instruction cycle. As a result, the code density of VLIW architectures may be low, if insufficient parallelism is detected to keep all functional units busy. The problem can be avoided if more flexibility is added. For example, the Texas Instruments TMS 320C6xx family of processors implements a variable instruction packet size of up to 256 bits. In each instruction field, one bit is reserved to indicate whether or not the operation encoded in the next field is still assumed to be executed in parallel (see fig. 3.27). No instruction bits are wasted for unused functional units.

![Figure 3.26. VLIW architecture (example)](image)

![Figure 3.27. Instruction packets for TMS 320C6xx](image)

Due to its variable length instruction packets, TMS 320C6xx processors do not quite correspond to the classical model of VLIW processors. Due to their explicit description of parallelism, they are EPIC processors, though.

### Partitioned Register Files

Implementing register files for VLIW and EPIC processors is far from trivial. Due to the large number of operations that can be performed in parallel, a large number of register accesses has to be provided in parallel. Therefore, a large number of ports is required. However, the delay, size and energy consumption of register files increases with their number of ports. Hence, register files with very large numbers of ports are inefficient. As a consequence, many VLIW/EPIC architectures use partitioned register files. Functional units are then only connected to a subset of the register files. As an example, fig. 3.28 shows the internal structure of the TMS 320C6xx processors. These processors...
have two register files and each of them is connected to half of the functional units. During each clock cycle, only a single path from one register file to the functional units connected to the other register file is available.

![Figure 3.28. Partitioned register files for TMS 320C6xx](image)

Alternative partitionings are considered by Lapinskii et al. [Lapinskii et al., 2001].

Many DSP processors are actually VLIW processors. As an example, we are considering the M3-DSP processor [Fettweis et al., 1998]. The M3-DSP processor is a VLIW processor containing (up to) 16 parallel data paths. These data paths are connected to a group memory, providing the necessary arguments in parallel (see fig. 3.29).

![Figure 3.29. M3-DSP (simplified)](image)

Predicated Execution

A potential problem of VLIW and EPIC architectures is their possibly large delay penalty: This delay penalty might originate from branch instructions found in some instruction packets. Instruction packets normally must pass through pipelines. Each stage of these pipelines implements only part of the operations to be performed by the instructions executed. The fact that branch instructions exist cannot be detected in the first stage of the pipeline. When the execution of the branch instruction is finally completed, additional instructions have already entered the pipeline (see fig. 3.30).

![Figure 3.30. Branch instruction and delay slots](image)

There are essentially two ways to deal with these additional instructions:

1. They are executed as if no branch had been present. This case is called delayed branch. Instruction packet slots that are still executed after a branch are called branch delay slots. These branch delay slots can be filled with instructions which would be executed before the branch if there were no delay slots. However, it is normally difficult to fill all delay slots with useful instructions and some must be filled with no-operation instructions (NOPs). The term branch delay penalty denotes the loss of performance resulting from these NOPs.

2. The pipeline is stalled until instructions from the branch target address have been fetched. There are no branch delay slots in this case. In this organization the branch delay penalty is caused by the stall.

Branch delay penalties can be significant. For example, the TMS 320C6xx family of processors has up to 40 delay slots. Therefore, efficiency can be improved by avoiding branches, if possible. In order to avoid branches originating from if-statements, predicated instructions have been introduced. For each predicated instruction, there is a predicate. This predicate is encoded in a few bits and evaluated at run-time. If the result is true, the instruction is executed. Otherwise, it is effectively turned into a NOP. Predication can also be found in RISC machines such as the ARM processor. Example: ARM instructions, as introduced on page 139, include a four-bit field. These four bits encode various expressions involving the condition code registers. Values stored in these
registers are checked at run-time. They determine whether or not a certain instruction has an effect.

Predication can be used to implement small if-statements efficiently: the condition is stored in one of the condition registers and if-statement-bodies are implemented as predicated instructions which depend on this condition. This way, if-statement bodies can be evaluated in parallel with other operations and no delay penalty is incurred.

The Crusoe processor is a (commercially finally unsuccessful) example of an EPIC processor designed for PCs [Klaiber, 2000]. Efforts for making EPIC instruction sets available in the PC sector resulted in Intel's IA-64 instruction set [Intel, 2010b] and its implementation in the Itanium® processor. Due to legacy problems, the main application is in the server market. Many MPSoCs (see page 151) are based on VLIW and EPIC processors.

3.3.3.4 Micro-controllers

A large number of the processors in embedded systems are in fact micro-controllers. Micro-controllers are typically not very complex and can be used easily. Due to their relevance for designing control systems, we introduce one of the most frequently used processors: the Intel 8051. This processor has the following characteristics:

- 8 bit CPU, optimized for control applications,
- large set of operations on Boolean data types,
- program address space of 64 k bytes,
- separate data address space of 64 k bytes,
- 4 k bytes of program memory on chip, 128 bytes of data memory on chip,
- 32 I/O lines, each of which can be addressed individually,
- 2 counters on the chip,
- universal asynchronous receiver/transmitter for serial lines available on the chip,
- clock generation on the chip,
- many variations commercially available.

All these characteristics are quite typical for micro-controllers.

3.3.3.5 Multiprocessor systems-on-a-chip (MPSoCs)

Further increase of clock rates of processors has recently come to a standstill. The large energy consumption of processors using multi-gigahertz clock speeds is a key reason for this. In order to still improve the overall performance, several processors must be employed. This led to the design of chips comprising multiple processors as well as additional components such as peripheral devices and memories. Systems implemented in that way are called MPSoCs (MultiProcessor System-on-a-Chip). For general purpose computing and PCs, multi-processor systems are typically homogeneous (all processors are of the same type). The term multi-core system is usually associated with such systems. For embedded systems, energy efficiency has top priority. Energy efficiency is typically obtained with highly specialized processors. For example, there may be specialized processors for mobile communication or image processing. Fig. 3.31 contains a simplified version of the floor-plan of the SH-MobileG1 chip [Hattori, 2007].

![Figure 3.31. Floor-plan of the SH-MobileG1 chip](image)

The chip demonstrates the fact, that highly specialized processors are being used: there are special processors for MPEG- and JPEG-encoding, for GSM- and 3G mobile communication etc. In order to save energy, unused areas are typically powered-down. Using such multi-processor-based systems from applications written in a sequential language is a challenge, which will be addressed in Chapter 6. Mapping techniques for such processors are important, since examples demonstrate that a power efficiency close to that of ASICs can be achieved. For example, for IMEC's ADRES processor, an efficiency of $55 \times 10^9$ operations per Watt (about 50% of the power efficiency of ASICs) has been predicted [Man, 2007], [IMEC, 2010].
### 3.3.4 Reconfigurable Logic

In many cases, full-custom hardware chips (ASICs) are too expensive and software-based solutions are too slow or too energy consuming. Reconfigurable logic provides a solution if algorithms can be efficiently implemented in custom hardware. It can be almost as fast as special-purpose hardware, but it is also more expensive. Reconfigurable logic can be changed by using configuration data. Due to these properties, reconfigurable logic finds applications in the following areas:

- **Fast prototyping**: modern ASICs can be very complex and the design effort can be large and takes a long time. It is therefore frequently desirable to generate a prototype, which can be used for experimenting with a system which behaves "almost" like the final system. The prototype can be more costly and larger than the final system. Also, its power consumption can be larger than the final system's power consumption.

- **Low volume applications**: if the expected market volume is too small to justify the development of special-purpose ASICs, reconfigurable logic can be the right hardware technology for applications, for which software would be too slow or too inefficient.

- **Real-time systems**: the timing of FPGA-based designs is typically known very precisely. Therefore, FPGAs can be used to implement timing-predictable systems.

Reconfigurable hardware frequently includes random access memory (RAM) to store configurations during normal operation of the hardware. Such RAM is normally volatile (the information is stored only while power is applied). Therefore, the configuration data must be copied into the configuration RAM at power-up. **Persistent storage technology** such as read-only memories (ROMs) and Flash memories will then provide the configuration data.

**Field programmable gate arrays** (FPGAs) are the most common form of reconfigurable hardware. As the name indicates, such devices are programmable "in the field" (after fabrication). Furthermore, they consist of arrays of processing elements. As an example, fig. 3.32 shows the array structure of Xilinx Virtex-II arrays [Xilinx, 2007].

The more recent Virtex-5 arrays contain up to 240 × 108 **configurable logic blocks** (CLBs) [Xilinx, 2009]. These can be connected using a programmable interconnect structure. Arrays also contain up to 1200 user input/output connections. In addition, there are up to 1056 DSP blocks comprising 25 × 18 bit multipliers and 16416 kbits of RAM (Block RAM). Each CLB consists of 2 so-called slices (see fig. 3.33).

![Floor-plan of Virtex-II FPGA](image)

**Figure 3.32**: Floor-plan of Virtex-II FPGA

![Virtex-5 CLB](image)

**Figure 3.33**: Virtex-5 CLB

Each slice contains four memories. Each memory can be used as a look-up table (LUT) for implementing a single 6-input logic function or two 5-input logic functions. All $2^6$ respectively all $2^5$ Boolean functions of 6 or 5 inputs can be implemented. With the help of multiplexers, several of these memories can be also be combined. Memories can also serve as ordinary RAM or as shift registers (SRLs). Each slice also includes four output registers and some special logic for fast additions (see fig. 3.34) [Xilinx, 2009].

Configuration data determines the setting of multiplexers in the slices, the clocking of registers and RAM, the content of RAM components and the con-
3.4 Memories

Data, programs, and FPGA configurations must be stored in some kind of memory. This must be done in an efficient way. Efficient means run-time, code-size and energy-efficient. Code-size efficiency requires a good compiler and can be improved with code compression (see page 138). Memory hierarchies can be exploited in order to achieve a good run-time and energy efficiency. The underlying reason is that large memories require more energy per access and are also slower than small memories.

Fig. 3.35 shows the cycle time and the power as a function of the size of memories used as register files [Rixner et al., 2000].

![Figure 3.35. Cycle time and power as a function of the register file size](image)

Power and delay for caches can be computed with CACTI [Wilton and Jouppi, 1996]. Generated values include the power and the delay for the data RAM. These values can be used to predict power and delay for general RAM memories. Fig. 3.36 shows the results for a larger range of sizes [Banakar et al., 2002].

![Figure 3.36. Power and delay of RAM memory as predicted by CACTI](image)

Connections between CLBs. Typically, this configuration data is generated from a high-level description of the functionality of the hardware, for example in VHDL. Ideally, the same description could also be used for generating ASICs automatically. In practice, some interaction is required.

Integration of reconfigurable computing with processors and software is simplified if processors are available in the FPGAs. There may be either hard cores or soft cores. For hard cores, the layout contains a special area implementing a core in a dense way. This area cannot be used for anything but the hard core. Soft cores are available as synthesizable models which are mapped to standard CLBs. Soft cores are more flexible, but less efficient than hard cores.

For example, the Virtex-5 FXT product line from Xilinx contains up to 2 Power-PC processors as hard cores.

Soft cores can be implemented on any FPGA chip. The MicroBlaze processor [Xilinx, 2008] is an example of such cores.
It has been observed that the difference in speeds between processors and memories is expected to increase (see fig. 3.37).

![Figure 3.37: Increasing gap between processor and memory speeds](image)

While the speed of memories is increasing by only a factor of about 1.07 per year, overall processor performance has increased by a factor of 1.5 to 2 per year [Machnik, 2002]. This means that the gap between processor performance and memory speeds is becoming larger. However, increasing processor performance further requires the use of multi-core systems.

Therefore, it is important to use smaller and faster memories that act as buffers between the main memory and the processor. In contrast to PC-like systems, the architecture of these small memories must guarantee a predictable real-time performance. A combination of small memories containing frequently used data and instructions and a larger memory containing the remaining data and instructions is generally more energy-efficient than a single, large memory. Memory partitioning has been considered, for example, by A. Macii [Macii et al., 2002].

Caches were initially introduced in order to provide good run-time efficiency. In the context of fig. 3.35 (right) however, it is obvious that caches potentially also improve the energy-efficiency of a memory system. Accesses to caches are accesses to small memories and therefore may require less energy per access than large memories. However, for caches it is required that the hardware checks whether or not the cache has a valid copy of the information associated with a certain address. This check involves comparing the tag fields of caches, containing a subset of the relevant address bits [Hennessy and Patterson, 2002]. Reading these tags requires additional energy. Also, the predictability of the real-time performance of caches is frequently low.

Alternatively, small memories can be mapped into the address space (see fig. 3.38).

![Figure 3.38: Memory map with scratch-pad included](image)

Such memories are called scratch pad memories (SPMs). Frequently used variables and instructions should be allocated to that address space and no checking needs to be done in hardware. As a result, the energy per access is reduced. Fig. 3.39 shows a comparison between the energy required per access to the scratch-pad (SPM) and the energy required per access to the cache.

![Figure 3.39: Energy consumption per scratch-pad and cache access](image)

For a two-way set associative cache, the two values differ by a factor of about three. The values in this example were computed using the energy consumption for RAM arrays as estimated by the CACTI cache estimation tool [Wilston and Jouppi, 1996].

SPMs can improve the memory access times very predictably, if the compiler is in charge of keeping frequently used variables in the SPM (see page 297).

### 3.5 Communication

Information must be available before it can be processed in an embedded system. Information can be communicated through various channels. Channels are abstract entities characterized by the essential properties of communication, like maximum information transfer capacity and noise parameters. The
probability of communication errors can be computed using communication theory techniques. The physical entities enabling communication are called communication media. Important media classes include: wireless media (radio frequency media, infrared), optical media (fibers), and wires.

There is a huge variety of communication requirements between the various classes of embedded systems. In general, connecting the different embedded hardware components is far from trivial. Some common requirements can be identified.

### 3.5.1 Requirements

The following list contains some of the requirements that must be met:

- **Real-time behavior:** This requirement has far-reaching consequences on the design of the communication system. Several low-cost solutions such as standard Ethernet fail to meet this requirement.

- **Efficiency:** Connecting different hardware components can be quite expensive. For example, point to point connections in large buildings are almost impossible. Also, it has been found that separate wires between control units and external devices in cars significantly add to the cost and the weight of the car. With separate wires, it is also very difficult to add new components. The need of providing cost-efficient designs also affects the way in which power is made available to external devices. There is frequently the need to use a central power supply in order to reduce the cost.

- **Appropriate bandwidth and communication delay:** Bandwidth requirements of embedded systems may vary. It is important to provide sufficient bandwidth without making the communication system too expensive.

- **Support for event-driven communication:** Polling-based systems provide a very predictable real-time behavior. However, their communication delay may be too large and there should be mechanisms for fast, event-oriented communication. For example, emergency situations should be communicated immediately and should not remain unnoticed until some central controller polls for messages.

- **Robustness:** Cyber-physical systems may be used at extreme temperatures, close to major sources of electromagnetic radiation etc. Car engines, for example, can be exposed to temperatures less than -20 and up to +180 degrees Celsius (-4 to 356 degrees Fahrenheit). Voltage levels and clock frequencies could be affected due to this large variation in temperatures. Still, reliable communication must be maintained.

- **Fault tolerance:** Despite all the efforts for robustness, faults may occur. Cyber-physical systems should be operational even after faults, if at all feasible. Restarts, like the ones found in personal computers, cannot be accepted. This means that retries may be required after attempts to communicate failed. A conflict exists with the first requirement: If we allow retries, then it is difficult to meet strict real-time requirements.

- **Maintainability, diagnosability:** Obviously, it should be possible to repair embedded systems within reasonable time frames.

- **Privacy:** Ensuring privacy of confidential information may require the use of encryption.

These communication requirements are a direct consequence of the general characteristics of embedded/cyber-physical systems mentioned in Chapter 1. Due to the conflicts between some of the requirements, compromises must be made. For example, there may be different communication modes: one high-bandwidth mode guaranteeing real-time behavior but no fault tolerance (this mode is appropriate for multimedia streams) and a second fault-tolerant, low-bandwidth mode for short messages that must not be dropped.

### 3.5.2 Electrical robustness

There are some basic techniques for electrical robustness. Digital communication within chips is normally using so-called single-ended signaling. For single-ended signaling, signals are propagated on a single wire (see fig. 3.40).

![Figure 3.40. Single-ended signaling](image)

Such signals are represented by voltages with respect to a common ground (less frequently by currents). A single ground wire is sufficient for a number of single-ended signals. Single ended signaling is very much susceptible to external noise. If external noise (originating from, for example, motors being switched on) affects the voltage, messages can easily be corrupted. Also, it is difficult to establish high-quality common ground signals between a large number of communicating systems, due to the resistance (and inductance) on the ground wires. This is different for differential signaling. For differential signaling, each signal needs two wires (see fig. 3.41).
Using differential signaling, binary values are encoded as follows: If the voltage on the first wire with respect to the second is positive, then this is decoded as '1', otherwise values are decoded as '0'. The two wires will typically be twisted to form so-called twisted pairs. There will be local ground signals, but a non-zero voltage between the local ground signals does not hurt. Advantages of differential signaling include:

- Noise is added to the two wires in essentially the same way. The comparator therefore removes almost all the noise.
- The logic value depends just on the polarity of the voltage between the two wires. The magnitude of the voltage can be affected by reflections or because of the resistance of the wires; this has no effect on the decoded value.
- Signals do not generate any currents on the ground wires. Hence, the quality of the ground wires becomes less important.
- No common ground wire is required. Hence, there is no need to establish a high quality ground wiring between a large number of communicating partners (this is one of the reasons for using differential signaling for Ethernet).
- As a consequence of the properties mentioned so far, differential signaling allows a larger throughput than single-ended signaling.

However, differential signaling requires two wires for every signal and it also requires negative voltages (unless it is based on complementary logic signals using voltages for single-ended signals).

Differential signaling is used, for example, in standard Ethernet-based networks.

### 3.5.3 Guaranteeing real-time behavior

For internal communication, computers may be using dedicated point-to-point communication or shared buses. Point-to-point communication can have a good real-time behavior, but requires many connections and there may be congestion at the receivers. Wiring is easier with common, shared buses. Typically, such buses use priority-based arbitration if several access requests to the communication media exists (see, for example, [Hennessy and Patterson, 2002]). Priority-based arbitration comes with poor timing predictability, since conflicts are difficult to anticipate at design time. Priority-based schemes can even lead to "starvation" (low-priority communication can be completely blocked by higher priority communication). In order to get around this problem, time division multiple access (TDMA) can be used. In a TDMA scheme, each partner is assigned a fixed time slot. The partner is allowed to transmit during that particular time slot. Typically, communication time is divided into frames. Each frame starts with some time slot for frame synchronization, and possibly some gap to allow the sender to turn off (see fig. 3.42, [Koopman and Upender, 1995]).

<table>
<thead>
<tr>
<th>Frame period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame sync</td>
</tr>
<tr>
<td>Frame gap</td>
</tr>
<tr>
<td>Slice 0</td>
</tr>
<tr>
<td>Slice 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Slice n-1</td>
</tr>
<tr>
<td>Frame sync</td>
</tr>
</tbody>
</table>

**Figure 3.42.** TDMA-based communication

This gap is followed by a number of slices, each of which serves for communicating messages. Each slice also contains some gap and guard time to take clock speed variations of the partners into account. Slices are assigned to communication partners. Variations of this scheme exist. For example, truncation of unused slices or the assignment of partners to several slides are feasible. TDMA reduces the maximum amount of data available per frame and partner, but guarantees a certain bandwidth for all partners. Starvation can be avoided. The ARM AMBA-bus [ARM Ltd., 2009a] includes TDMA-based bus allocation.

Communication between computers is frequently based on Ethernet standards. For 10 Mbit/s and 100 Mbit/s versions of Ethernet, there can be collisions between various communication partners. This means: several partners are trying to communicate at about the same time and the signals on the wires are corrupted. Whenever this occurs, the partners must stop communications, wait
for some time, and then retry. The waiting time is chosen at random, so that it is not very likely that the next attempt to communicate results in another collision. This method is called **carrier-sense multiple access/collision detect** (CSMA/CD). For CSMA/CD, communication time can get huge, since conflicts can repeat a large number of times, even though this is not very likely. Hence, CSMA/CD cannot be used when real-time constraints must be met.

This problem can be solved with CSMA/CA (**carrier-sense multiple access/collision avoidance**). As the name indicates, collisions are completely avoided, rather than just detected. For CSMA/CA, priorities are assigned to all partners. Communication media are allocated to communication partners during **arbitration phases**, which follow communication phases. During arbitration phases, partners wanting to communicate indicate this on the media. Partners finding such indications of higher priority must immediately remove their indication.

Provided that there is an upper bound on the time between arbitration phases, CSMA/CA guarantees a predictable real-time behavior for the partner having the highest priority. For other partners, real-time behavior can be guaranteed if the higher priority partners do not continuously request access to the media.

Note that high-speed versions of Ethernet (≥ 1 Gbit/s) also avoid collisions. TDMA-schemes are also used for wireless communication. For example, mobile phone standards like GSM use TDMA for accesses to the communication medium.

### 3.5.4 Examples

- **Sensor/actuator buses**: Sensor/actuator buses provide communication between simple devices such as switches or lamps and the processing equipment. There may be many such devices and the cost of the wiring needs special attention for such buses.

- **Field buses**: Field buses are similar to sensor/actuator buses. In general, they are supposed to support larger data rates than sensor/actuator buses. Examples of field buses include the following:
  - **Controller Area Network (CAN)**: This bus was developed in 1981 by Bosch and Intel for connecting controllers and peripherals. It is popular in the automotive industry, since it allows the replacement of a large amount of wires by a single bus. Due to the size of the automotive market, CAN components are relatively cheap and are therefore also used in other areas such as smart homes and fabrication equipment. CAN has the following properties:
    * differential signaling with twisted pairs,
    * arbitration using CSMA/CA,
    * throughput between 10kbit/s and 1 Mbit/s,
    * low and high-priority signals,
    * maximum latency of 134 μs for high priority signals,
    * coding of signals similar to that of serial (RS-232) lines of PCs, with modifications for differential signaling.

CSMA/CA-based arbitration does not prevent starvation. This is an inherent problem of the CAN protocol.


- **FlexRay** [FlexRay Consortium, 2002] is a TDMA protocol which has been developed by the FlexRay consortium (BMW, DaimlerChrysler, General Motors, Ford, Bosch, Motorola and Philips Semiconductors). FlexRay is a combination of a variant of the TTP and the byteflight [Byteflight Consortium, 2003] protocol.

FlexRay includes a static as well as a dynamic arbitration phase. The static phase uses a TDMA-like arbitration scheme. It can be used for real-time communication and starvation can be avoided. The dynamic phase provides a good bandwidth for non-real-time communication. Communicating partners can be connected to up to two buses for fault-tolerance reasons. **Bus guardians** may protect partners against partners flooding the bus with redundant messages, so-called **babbling idiots**. Partners may be using their own local clock periods. Periods common to all partners are defined as multiples of such local clock periods. Time slots allocated to partners for communication are based on these common periods.

The Levi simulation allows simulating the protocol in a lab environment [Sirovic and Marwedel, 2007a].

- **LIN** (Local Interconnect Network) is a low-cost communication standard for connecting sensors and actuators in the automotive domain [LIN Administration, 2010].

- **MAP**: MAP is a bus designed for car factories.

- **EIB**: The European Installation Bus (EIB) is a bus designed for smart homes.

- **Wired multimedia communication**: For wired multimedia communication, larger data rates are required. Example: **MOST** (Media Oriented Systems Transport) is a communication standard for multimedia and infotainment equipment in the automotive domain [MOST Cooperation, 2010]. Standards like IEEE 1394 (FireWire) may be used for the same purpose.
Wireless communication: This kind of communication is becoming more popular. Currently (2010), 7 Mbit/s are widely available with HSPA (High Speed Packet Access). Even higher rates (based, for example, on the long-term evolution (LTE) technology) are on the horizon. Bluetooth is a standard for connecting devices such as mobile phones and their headsets.

The wireless version of Ethernet is standardized as IEEE standard 802.11. It is being used in local area networks (LANs).

DECT is a standard used for wireless phones in Europe.

3.6 Output

Output devices of embedded/cyber-physical systems include:

- **Displays**: Display technology is an area which is extremely important. Accordingly, a large amount of information [Society for Display Technology, 2003] exists on this technology. Major research and development efforts lead to new display technology such as organic displays [Gelsen, 2003]. Organic displays are emitting light and can be fabricated with very high densities. In contrast to LCD displays, they do not need back-light and polarizing filters. Major changes are therefore expected in these markets.

- **Electro-mechanical devices**: these influence the environment through motors and other electro-mechanical equipment.

Analog as well as digital output devices are used. In the case of analog output devices, the digital information must first be converted by digital-to-analog (D/A)-converters. These converters can be found on the path from analog inputs of embedded systems to their outputs. Fig. 3.43 shows the naming convention of signals along the path which we use. Purpose and function of the boxes will be explained in this section.

![Figure 3.43. Naming convention for signals between analog inputs and outputs](image)

3.6.1 D/A-converters

D/A-converters are not very complex. Fig. 3.44 shows the schematic of a simple so-called weighted-resistor D/A converter.

![Figure 3.44. D/A-converter](image)

The key idea of the converter is to first generate a current which is proportional to the value represented by a digital signal $x$. Such a current can hardly be used by a following system. Therefore, this current is converted into a proportional voltage $y$. This conversion is done with an operational amplifier (depicted by a triangle in fig. 3.44). Essential characteristics of operational amplifiers are described in Appendix B of this book.

How do we compute the output voltage $y$? Let us first consider the loop indicated by the dashed line in fig. 3.44. The current through any resistor is zero, if the corresponding element of digital signal $x$ is 0. If it is 1, the current corresponds to the weight of that bit, since resistor values are chosen accordingly. We can apply Kirchhoff’s Loop Rule (see Appendix B) to the loop turned on by the least significant bit $x_0$ of $x$. We have

$$x_0 \cdot I_0 - \frac{V_+}{8} + V_{ref} = 0 \quad (3.17)$$

$V_+$ is approximately 0 (see Appendix B). Therefore, we have

$$I_0 = \frac{x_0 \cdot V_{ref}}{8 + R} \quad (3.18)$$

Corresponding equations hold for the currents $I_1$ to $I_5$ through the other resistors. We can now apply Kirchhoff’s Node Rule (see Appendix B) to the circuit node connecting all resistors. At this node, the outgoing current must be equal to the sum of the incoming currents. Therefore, we have

$$I = I_1 + I_2 + I_3 + I_4 + I_0 \quad (3.19)$$

$$I = \frac{x_3 \cdot V_{ref}}{R} + \frac{x_2 \cdot V_{ref}}{2 \cdot R} + \frac{x_1 \cdot V_{ref}}{4 \cdot R} + x_0 \cdot \frac{V_{ref}}{8 \cdot R}$$
\begin{equation}
\frac{V_{ref}}{R} \sum_{i=0}^{3} x_i \cdot 2^{-i} \cdot 3
\end{equation}

Now, we can apply Kirchhoff's Loop Rule to the loop comprising \( R_1 \), \( y \) and \( V_\cdot \). Since \( V_\cdot \) is approximately 0, we have:

\begin{equation}
y + R_1 \cdot I = 0.
\end{equation}

Next, we can apply Kirchhoff's Node Rule to the node connecting \( I \), \( I' \) and the inverting signal input of the operational amplifier. The current into this input is practically zero, and currents \( I \) and \( I' \) are equal: \( I = I' \). Hence, we have:

\begin{equation}
y + R_1 \cdot I = 0
\end{equation}

From equations 3.20 and 3.22 we obtain:

\begin{equation}
y = -V_{ref} \cdot \frac{R_1}{R} \sum_{i=0}^{3} x_i \cdot 2^{-i} \cdot 3 = -V_{ref} \cdot \frac{R_1}{8 \cdot R} \cdot \text{nat}(x)
\end{equation}

\text{nat} denotes the natural number represented by digital signal \( x \). Obviously, \( y \) is proportional to the value represented by \( x \). Positive output voltages and bit-vectors representing two's complement numbers require minor extensions.

From a DSP point of view, \( y(t) \) is a function over a discrete time domain: it provides us with a sequence of voltage levels. In our running example, it is defined only over integer times. From a practical point of view, this is inconvenient, since we would typically observe the output of the circuit of fig. 3.44 continuously. Therefore, D/A-converters are frequently extended by a "zero-order hold" functionality. This means that the converter will keep the previous value until the next value is converted. Actually, the D/A-converted of fig. 3.44 will do exactly this if we do not change the settings of the switches until the next discrete time instant. Hence, the output of the converter is a step function \( y'(t) \) corresponding to the sequence \( y(t) \). \( y'(t) \) is a function over the continuous time domain.

As an example, let us consider the output resulting from the conversion of the signal of equation 3.3, assuming a resolution of 8 steps per polarity. For this case, fig. 3.45 shows \( y'(t) \) instead of \( y(t) \), since \( y'(t) \) is a bit easier to visualize.

**Figure 3.45.** Step function \( y'(t) \) generated from signal \( e(t) \) (eq. 3.3) sampled at integer times.

D/A-converters enable a conversion from time- and value-discrete signals to signals in the continuous time and value domain. However, neither \( y(t) \) nor \( y'(t) \) reflect the values of the input signal in-between the sampling instances.

**3.6.2 Sampling theorem**

Suppose that the processors used in the hardware loop forward values from A/D-converters uncharged to the D/A-converters. We could also think of storing values \( x(t) \) on a CD and aiming at generating an excellent analog audio signal. Would it be possible to reconstruct the original analog voltage \( e(t) \) (see fig. 3.8, fig. 3.20, and fig. 3.43) at the outputs of the D/A-converters?

It is obvious that reconstruction is not possible if we have aliasing of the type described in the section on sampling\(^\text{10}\). So, we assume that the sampling rate is larger than twice the highest frequency of the decomposition of the input signal into sine waves (sampling criterion, see equation 3.8). Does meeting this criterion allow us to reconstruct the original signal? Let us have a closer look!

Feeding D/A-converters with a discrete sequence of digital values will result in a sequence of analog values being generated. Values of the input signal in-between the sampling instances are not generated by D/A-converters. The simple zero-order hold functionality (if present) would generate only step functions. This seems to indicate that reconstruction of \( e(t) \) would require an infinitely large sampling rate, such that all intermediate values can be generated.

\(^{10}\)Reconstruction may be possible, if additional information about the signal is available, i.e. if we restrict ourselves to certain signal types.
However, there could be some kind of smart interpolation computing values in-between the sampling instances from the values at sampling instances. And indeed, sampling theory [Oppenheim et al., 2009] tells us that a corresponding time-continuous signal $z(t)$ can be constructed from the sequence $y(t)$ of analog values.

Let $\{t_s\}, s = ..., -1, 0, 1, 2, ...$ be the times at which we sample our input signal. Let us assume a constant sampling rate of $f_s = \frac{1}{p_s}$ (\(\forall s: p_s = t_{s+1} - t_s\)). Then, sampling theory tells us that we can approximate $c(t)$ from $y(t)$ as follows:

$$z(t) = \sum_{s=-\infty}^{\infty} y(t_s) \frac{\sin \frac{\pi}{p_s}(t-t_s)}{\pi \frac{p_s}{p_s}(t-t_s)}$$

(3.24)

This equation is known as the Shannon-Whittaker interpolation. $y(t_s)$ is the contribution of signal $y$ at sampling instance $t_s$. The influence of this contribution decreases the further $t$ is away from $t_s$. The decrease follows a weighting factor, also known as the sinc function:

$$sinc(t-t_s) = \frac{\sin \frac{\pi}{p_s}(t-t_s)}{\frac{\pi}{p_s}(t-t_s)}$$

(3.25)

which decreases non-monotonically as a function of $|t-t_s|$. This weighting factor is used to compute values in-between the sampling instances. Fig. 3.46 shows the weighting factor for the case $p_s = 1$.

At each of the sampling instances $t_s$ (integer times in our case), $z(t_s)$ is computed just from the corresponding value $y(t_s)$, since the sinc function is zero in this case for all other sampled values. In between the sampling instances, all of the adjacent discrete values contribute to the resulting value of $z(t)$. Fig. 3.49 shows the resulting $z(t)$ if $e(t) = e(t)$ and processing performs the identity function ($x(t) = w(t)$).
The figure includes signals $e_3(t)$ (solid line), $z(t)$ (dotted line), and $y'(t)$ (dashed line). $z(t)$ is based on summing up the contributions of all sampling instances shown in the diagrams 3.47 and 3.48. $e_3(t)$ and $z(t)$ are very similar.

How close could we get to the original input signal by implementing equation 3.24? Sampling theory tells us (see, for example, [Oppenheim et al., 2009]), that \textit{equation 3.24 computes an exact approximation}, if the sampling criterion (equation 3.8) is met. Therefore, let us see how we can implement equation 3.24.

How do we compute equation 3.24 in an electronic system? We cannot compute this equation in the discrete time domain using a digital signal processor for this, since this computation has to generate a time-continuous signal. Computing such a complex equation with analog circuits seems to be difficult at first sight.

Fortunately, the required computation is a so-called \textit{folding operation} between signal $y(t)$ and the \textit{sinc}-function. According to the classical theory of Fourier transforms, a folding operation in the time domain is equivalent to a multiplication with frequency-dependent filter function in the frequency domain. This filter function is the Fourier transform of the corresponding function in the time domain. Therefore, equation 3.24 can be computed with some appropriate filter. Fig. 3.50 shows the corresponding placement of the filter.

The remaining question is: which frequency-dependent filter function is the Fourier transform of the \textit{sinc}-function? Computing the Fourier transform of the \textit{sinc}-function yields a low-pass filter function [Oppenheim et al., 2009]. So, “all” we must do to compute equation 3.24 is to pass signal $y(t)$ through a low-pass filter, filtering frequencies as shown for the “ideal filter” in fig. 3.51. Note that the representation of function $y(t)$ as a sum of sine waves would require very high frequency components, making such a filtering non-redundant, even though we have already assumed an anti-aliasing filter to be present at the input.

There is still one problem, though: ideal low-pass filters do not exist. Therefore, we must live with compromises and design filters approximating the low pass filter characteristics. Actually, we must live with several imperfections preventing a precise reconstruction of the input signals:

- Ideal low pass filters cannot be designed. Therefore, we must use approximations of such filters. Designing good compromises is an art (performed extensively, for example, for audio equipment).
- For the same reason, we cannot completely remove input frequencies beyond the Nyquist frequency.
- The impact of value quantization is visible in fig. 3.49. Due to value quantization, $e_3(t)$ is sometimes different from $z(t)$. Quantization noise, as introduced by A/D-converters, cannot be removed during output generation. Signal $w(t)$ from the output of the A/D-converter will remain distorted by the quantization noise. However, this effect does not affect the signal $h(t)$ from the output of sample-and-hold circuits.
- Equation 3.24 is based on an infinite sum, involving also values at future instances in time. In practice, we can delay signals by some finite amount.
to know a finite number of “future” samples. Infinite delays are impossible. In fig. 3.49, we did not consider contributions of sampling instances outside the diagram.

The functionality provided by low-pass filters demonstrates the power of analog circuits: there would be no way of implementing the behavior of analog filters in the digital domain, due to the inherent restriction to discretized time and values.

Many authors have contributed to sampling theory. Therefore, many names can be associated with the sampling theorem. Contributors include Shannon, Whittaker, Kotelnikov, Nyquist, Körpfmüller, and others. Therefore, the fact that the original signal can be reconstructed should simply be called the sampling theorem, since there is no way of attaching all names of relevant contributors to the theorem.

3.6.3 Actuators

There is a huge amount of actuators [Elsevier B.V., 2010a]. Actuators range from huge ones that are able to move tons of weight to tiny ones with dimensions in the μm area, like the one shown in fig. 3.52.

![Image](image_url)

**Figure 3.52.** Microsystem technology based actuator motor (partial view; courtesy E. Obemeier, MAT, TU Berlin), ©TU Berlin

It is impossible to provide a complete overview. As an example, we mention only a special kind of actuators which will become more important in the future: microsystem technology enables the fabrication of tiny actuators, which can be put into the human body, for example.

Using such tiny actuators, the amount of drugs fed into the body can be adapted to the actual need. This allows a much better medication than needle-based injections. Fig. 3.52 shows a tiny motor manufactured with microsystem technology. The dimensions are in the μm range. The rotating center is controlled by electrostatic forces.

3.7 Secure hardware

The general requirements for embedded systems can often include security (see page 5). If security is a major concern, special secure hardware may need to be developed. Security may need to be guaranteed for communication and for storage [Krivovjak and Matyas, 2006]. Also, security might demand special equipment for the generation of cryptographic keys. Special hardware security modules have been designed. One of the goals for such modules is to resist side-channel attacks such as measurement of the supply current or electromagnetic radiation. Such modules include special mechanisms for physical protection (shielding, or sensors to detect tampering with the modules). Special processors may support encryption and decryption. In addition to the physical security, we need logical security, typically using cryptographic methods. Smart cards are a special case of secure hardware that must run using a very small amount of energy. In general, it is necessary to distinguish between different levels of security and levels of knowledge of “adversaries”. A full presentation of the techniques for designing secure hardware is beyond the scope of this book. Interested readers are referred to Gebotys [Gebotys, 2010] and workshop proceedings [Clavier and Gaj, 2009].

3.8 Assignments

1. It is suggested that locally available small robots are used to demonstrate hardware in the loop, corresponding to fig. 3.2. The robots should include sensors and actuators. Robots should run a program implementing a control loop. For example, an optical sensor could be used to let a robot follow a black line on the ground. The details of this assignment depend on the availability of robots.

2. Why is it so important to optimize embedded systems? Compare different technologies for processing information in an embedded system with respect to their efficiency!

3. Assume that we have an input signal x consisting of the sum of sine waves of 1.75 kHz and 2 kHz. We are sampling x at a rate of 3 kHz. Will we be able to reconstruct the original signal after discretization of time? Please explain your result!

4. Discretization of values is based on A/D-converters. Develop the schematic of a flash-based A/D-converter for positive and negative input voltages!
The output should be encoded as 3-bit two’s complement numbers, allowing to distinguish between 8 different voltage intervals.

5 Compare the complexity of flash-based and successive approximation-based A/D-converters. Assume that you would like to distinguish between $n$ different voltage intervals. Enter the complexity into the table of fig. 3.53, using the $O$-notation.

<table>
<thead>
<tr>
<th>Time complexity</th>
<th>Successive approximation converter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.53. Complexity of A/D-converters

6 Suppose that we are working with a successive approximation-based 4-bit A/D-converter. The input voltage range extends from $V_{\text{min}} = 1 \text{ V} (=0000^*)$ to $V_{\text{max}} = 4.75 \text{ V} (=1111^*)$. Which steps are used to convert voltages of 2.25 V, 3.75 V, and 1.8 V? Draw a diagram similar to fig. 3.12 which depicts the successive approximation to these voltages!

7 Extend the flash-based A/D converter such that it can be used for negative voltages as well!

8 Suppose a sine wave is used as an input signal to the converter designed in assignment 4. Depict the quantization noise signal for this case!

9 Create a list of features of DSP-processors!

10 Which components do FPGA comprise? Which of these are used to implement Boolean function? How are FPGAs configured? Are FPGAs energy-efficient? Which kind of applications are FPGAs good for?

11 In the context of memories, we are sometimes saying “small is beautiful”. What could be the reason for this?

12 Develop the following FlexRay\textsuperscript{TM} cluster: The cluster consists of the 5 nodes A, B, C, D and E. All nodes should be connected via two channels. The cluster uses a bus topology. The nodes A, B and C are executing a safety critical task and therefore their bus requests should be guaranteed at the time of 20 macroticks. The following is expected from you:

- Download the levi FlexRay simulator [Sirocic and Marwedel, 2007a].
- Unpack the .zip file and install!
- Start the training module by executing the file leviFRP.jar.
- Design the described FlexRay cluster within the training module.

- Configure the communication cycle such that the nodes A, B and C have a guaranteed bus access within a maximal delay of 20 macroticks. The nodes D and E should use only the dynamic segment.

- Configure the node bus requests. The node A sends a message every cycle. The nodes B and C send a message every second cycle. The node D sends a message of the length of 2 minislots every cycle and the node E sends every second cycle a message of the length of 2 minislots.

- Start the visualization and check if the bus requests of the nodes A, B and C are guaranteed.

- Swap the positions of nodes D and E in the dynamic segment. What is the resulting behavior?

13 Develop the schematic of a 3-bit D/A-converter! The conversion should be done for a 3-bit vector $x$ encoding positive numbers. Prove that the output voltage is proportional to the value represented by the input vector $x$. How would you modify the circuit if $x$ represented two’s complement numbers?

14 The circuit shown in fig. B.4 in Appendix B is an amplifier, amplifying input voltage $V_1$:

$$V_{\text{out}} = g_{\text{closed}} \cdot V_1$$

Compute the gain $g_{\text{closed}}$ for the circuit of fig. B.4 as a function of $R$ and $R_1$!