# 1

# Real-Time Systems and Time Predictability

This introduction discusses real-time systems and shows how they differ from general purpose computing systems. Different types of real-time systems are explained and examples of these types are given. The need for time predictability of computer systems is exposed and the structure of the book is presented.



## 1.1. Real-time systems

### 1.1.1. *Introduction*

Most of the processors in use today are not part of what we usually call *computers*, i.e. servers, workstations, laptops or tablets. They are instead components of hidden computing systems that control and interact with physical environments, also referred to as *embedded systems*. Such systems are developed in a wide range of domains: transport (cars, aircrafts, trains and rockets), household appliances (washing machines and vacuum cleaners), communications (cellular phones, modems and routers), multimedia (MP3 players and set top boxes), construction machinery (drilling machines), production lines (robots), medicine (pacemakers), etc. In cars, for example, embedded software controls the behavior of the engine with the goal of saving fuel and, at the same time, limiting emissions. Digital equipment also provides safety improvement by the means of dedicated functions (e.g. antilock breaking systems and air bags) and the well-being of passengers (e.g. air conditioning, power windows or audio systems). In recent years, the notion of *cyber-physical systems* has been introduced. This refers to such

systems that link several computing elements, which tightly interact both among one another and with their physical environment.

Embedded systems typically abide by a number of constraints that go beyond the usual scope of computing systems. Economical considerations, especially for large markets such as those for phones or cars, impose optimizing the cost. This is particularly true for hardware components: even an additional 5 cents is a lot when multiplied by 1,000,000 units. But cost issues also concern software development. For this reason, the reuse of software components (intellectual property (IP) components) is usually favored. Hand-held electronic devices should be as small and as light as possible. But size and, above all, weight may also be an issue in some transportation systems: additional weight usually increases the cost. Mobile devices must exhibit low energy consumption to optimize the life of batteries between charges. Thermal dissipation may be an issue in cases of limited cooling facilities, e.g. in confined spaces. Other constraints are put on systems that operate in harsh environments and are subjected to heat, vibrations, shocks, radio frequency (RF) interference, corrosion water, etc. Components in aerospace systems can be hit by cosmic and high-energy ionizing particles that engender the so-called single event upset that may change the state of bits. Radiation-hardened components are used in spatial systems but also around nuclear reactors.

Furthermore, some embedded systems must fulfill timing constraints: some of the tasks that implement the system must meet deadlines. For such systems, referred to as *real-time systems*, producing results in time is required as much as producing correct results. As a result, part of the system design and verification process consists of performing timing analysis of critical tasks, then checking that they can be scheduled in such a way that they are able to meet their deadlines. Various techniques and tools have been developed for this purpose and will be surveyed throughout this book. In section 1.1.2, the concept of task criticality will be discussed, safety standards will be briefly reviewed in section 1.1.3 and various examples of real-life real-time systems will be discussed in section 1.1.4.

Progress in technology and in computer architecture designs leads to components that offer steadily increasing computing power. Today, increasing clock frequencies in order to obtain higher performance from advanced processors is no longer feasible. So, the trend is to integrate multiple cores on

a single chip: the aggregated performance is higher than what can be achieved with single-core architectures, and at the same time the performance/energy ratio is improved. Better performance allows us to consider the implementation of new and advanced functionalities, such as steer-by-wire and brake-by-wire driver-assistance systems, combustion engine control, or automatic emergency-braking tiggered by collision avoidance techniques. These systems can evaluate more sensor signals and master more complex situations if higher performance is provided by future control units. Examples could be online distance measures that trigger higher security actions for passengers such as closing the windows and setting the passenger seats in upright positions if an unavoidable crash is detected. Motor injection could be optimized to reduce gas consumption and emissions through better processor performances. Embedded systems for automotive applications are under permanent pressure of cost, while demanding higher performance due to new standards and Quality-of-Service (QoS). Also of paramount and vital importance is the ability to develop and produce systems that are capable of achieving maximum safety, reliability and availability.

Aerospace applications benefit from more powerful processors by providing support for ever-increasing demands of additional functionality on board or a higher level of comfort through a better control of the actuators and, at the same time, requiring absolute guarantees on the timing performance of the system. In addition, certification requirements (e.g. DO-178C) impose restrictions on proving properties of hardware and software, especially regarding timing. At the very least, a fourfold increase in performance is desired for next-generation aircrafts.

On a similar level, future space applications will require increased performance; however, higher CPU clock frequency rates are not generally feasible due to the increased risk of electromagnetic interference and errors induced from cosmic radiation. In this context, an increase in performance is not due to increasing CPU frequencies, but increasing the number of computation units. The most energy and weight efficient way to achieve this is through multicore processors. Current trends in the development of the freely available LEON family of processors for space applications precisely follow this trend. However, in a project performed for ESA by Rapita Systems called PEAL, it was shown that there are still significant issues regarding the adoption of these advanced features by industry unless provable properties regarding timing can be demonstrated. In aerospace, the demand

for reduced weight and size is relentless. Guidance, navigation and control algorithms routinely coexist on a single processor. With the advent of free flight and autonomous flight, we must increasingly co-host many other safety and non-safety critical applications on a common powerful processor to maximally utilize hardware that reduces recurring costs, size, weight and power.

This race for computing performance has consequences on the verification and validation of critical systems. Most schemes implemented in modern processors to achieve high performance exhibit one of the following characteristics: (1) their behavior relies on the execution history (e.g. dynamic branch predictors make their decisions based on the issues of previous branches; cache memories contain instructions and data that have been accessed in the past); (2) their behavior is dynamic, i.e. it depends on information that is only available at runtime (e.g. the way an instruction crosses a pipeline depends on which instructions are in the pipeline at the same time); (3) they speculate on the results of some instructions in order to process other instructions faster. These characteristics combined with a wide range of possible values of input data make the execution profile of tasks difficult to predict at analysis time. With multicore architectures, the sharing of resources (e.g. the interconnection network and part of the memory) among cores adds to the complexity. Paradoxically, a faster architecture does not systematically mean a better chance of meeting deadlines: the sophisticated mechanisms used to increase the instruction rate are often hard, sometimes even impossible, to model, and this may result in longer estimated execution times due to the pessimism engendered by overcautious assumptions. In section 1.2, we review the concept of *time predictability*. One objective of this book is to show why some of the above-mentioned schemes challenge timing analysis techniques and to provide some recommendations for time-predictable architectures. Shortly, the global advice is: *Make the worst case fast and the whole system easy to analyze* [SCH 09b].

### 1.1.2. *Soft, firm and hard real-time systems*

Real-time systems are commonly divided into the following three categories according to the consequence of missing a deadline (see Figure 1.1):

– In *hard* real-time systems, missing a deadline is a full system failure. Dramatic consequences include environmental disaster, economic crash, or

even loss of human lives. An example of such a system is the control unit that triggers inflating air bags during a car crash. The decision to use the air bag is taken from monitoring various sensors, e.g. accelerometers, wheel speed sensors, gyroscopes, seat occupancy sensors and brake pressure sensors. It must be taken in time, i.e. within a given delay after the collision is anticipated, so that the driver and the passengers are protected against hitting the steering wheel and windows. Many other applications in the domain of traveler transportation (e.g. flight control software in an aircraft, signaling systems in trains or antilock braking systems (ABSs) in cars) have strict deadlines. An antimissile system also runs hard real-time tasks: it must destroy any incoming missile before it cause any damage.

– *Firm* real-time systems denote such systems for which a result produced after the deadline is useless and discarded without any catastrophic consequences. It is accepted that some deadlines can be missed if not too frequently. Typical firm real-time systems are encountered in the domain of multimedia applications. In a video conferencing system, audio or video frames that do not reach their destination before their deadline are simply dropped, which may only affect the quality of the received video.

– In *soft* real-time systems, the result produced by a task after its deadline is still useful but the QoS is degraded. Video decoding applications and printer control software belong to this category.

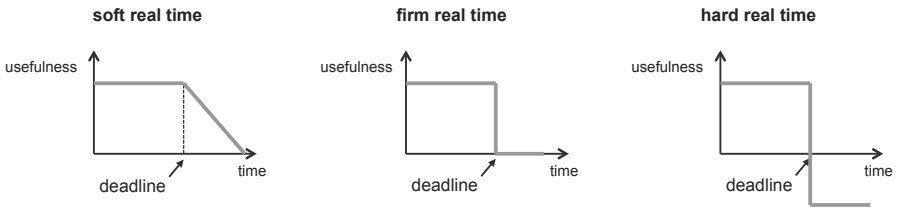


**Figure 1.1.** *Soft, firm and hard real time*

This book mainly deals with hard real-time systems where it is expected that no failure occurs during the time the system is effective (often referred to as the time of the mission in reference to aeronautics and space systems). Quantitatively, the probability of a hazard occuring, per operational hour, should be less than $10^{-9}$ [KAS 12]. Achieving such a low probability of failure requires a careful design of the hardware, taking into account the

specific constraints of the application domain, the use of fault-tolerance techniques and a normalized process for the development of the software including formal methods to ensure that the system is logically and timely error-prone.

The notion of *criticality* is closely related to the application domain. Transportation systems (airplanes, cars, trains and spacecraft) include critical subsystems for which a failure may translate into injuries to or even death of passengers. This is also true for equipment that may look light at first glance but that requires a high level of reliability, such as amusement park rides. A failure in the control system of a nuclear plant may engender an environmental catastrophe that can also have severe consequences on the health of neighboring people. Some medical devices, such as heart pacemakers, feature safety requirements as well, for obvious reasons. Now, criticality can also be considered through the prism of economical consequences. A failure in an electricity distribution system can put some industries in a difficult situation. A company that produces a component that produces timing errors may suffer economical effects and close if they lose markets.

### 1.1.3. *Safety standards*

Safety standards exist for several of the application domains mentioned above and their use spreads rapidly [KAS 12]. Their objective is to provide guidelines toward safe software development and validation. Besides guaranteeing the absence of non-functional hazards, they require verifying three points: absence of runtime errors, execution times and memory usage. They define criticality levels and recommend, for each level, a range of techniques and tools to be used to show that the software meets the constraints.

The DO-178C is applicable to aerospace systems. It defines five levels of criticality, from level A (the most critical) to level E (the least critical). The standard promotes software verification through the use of formal techniques, such as abstract interpretation, model checking and theorem proving (it explicitly mentions that dynamic testing “cannot show the absence of errors”). It also recommends model-based software development as well as the use of qualified tools. Worst-case execution time (WCET) is listed as one

of the non-functional properties that must be verified. The safety standard for electrical, electronic and programmable electronic systems is the IEC-61508. It considers four safety integrity levels, from SIL1 (the least critical) to SIL4 (the most critical). WCET is also mentioned as a property to be checked and static program analysis is required for levels SIL2–SIL4. The ISO-26262 is for automotive systems and the CENELEC EN-50128 is for railway systems. They both impose analyzing WCETs and response times and recommend static analysis techniques for this purpose. Similar standards exist for medical applications (the EN-60601 and the IEC-62304) and for nuclear plants (IEC-60880).

### 1.1.4. *Examples*

This section presents several examples of soft and hard real-time systems. Firm real-time systems are not included because of their marginal relevance in practice.

#### 1.1.4.1. *Soft real-time systems*

Soft real-time systems mainly appear in the domain of multimedia and software-defined radio systems. In general, multimedia systems translate a kind of digital data stream into visual or audio signals or vice versa. The visual or audio signals arise from or are consumed by human beings. This is where the real-time constraints come from. The reasons for the real-time constraints are the visual percipiency of human beings and the physical characteristics of audio signals.

##### 1.1.4.1.1. MP3 decoding

The so-called MP3 format for audio data (see [PAN 96]), which was originally called Moving Picture Expert Group (MPEG) layer 3, stores coded data of an audio stream. Several bit resolutions and sample rates are possible as well as the availability of mono- and stereo features.

The audio information stored within an MP3 stream is separated into multiple frames containing 32 or 12 samples, according to the wave band. Depending on the sample rate, each frame correlates to a predefined period of the audio stream. To obtain a proper playback of the MP3 audio stream, these frames must be read, decoded and handed to the output device in time.

Otherwise, a blackout time between the previous frame and the current frame occurs that can be noticed by the listener.

The observed blackout time is a result of a missed deadline, which is provided by the frame length. The execution time of the task that decodes the MP3 stream depends on the data present within the stream and some environmental activities. For example, the user of an MP3 player presses a button on the device while the player is decoding a frame. The pressed button should be handled within a short period of time otherwise the player seems to hang, but the handling could affect the decoding task and thus its execution time.

In the case of this example, a missed deadline results in an unintended blackout but no danger for human beings or machinery can occur. Hence, this example has to be classified as a soft real-time system.

1.1.4.1.2. MPEG decoding

Another example of a soft real-time system is the MPEG video decoding [138 96], which is similar to the MP3 decoding. In contrast to the MP3 coding, the MPEG video streams consist of different kinds of frames. The types of frames have different characteristics with respect to their complexity. The main types are the $I$, the $P$ and the $B$ frames. The $I$-type frames contain a whole picture with the full resolution of the video, whereas the two other types are composed of relative pictures, i.e. only deltas are stored within these frames.

During playback, a typical series of frames is $I, B, B, P, B, B, I$ where the $P$ is a delta with respect to the first $I$ and the $B$ frames are related to the preceding $I$ and synchronously to the succeeding $P$ frame and vice versa, respectively. As a result, the $P$ frame has to be decoded before the second frame can be shown. To allow for this circumstance, the frames are stored in a different order within the data stream, e.g. $I, P, B, B, I, B, B$. Hence, it is required to decode two frames, the $P$ and the first $B$, to display the second picture (after the initial $I$ frame). Accordingly, it is not required to decode any frame to display the fourth picture, which is the already-decoded $P$ frame.

Because of the different kinds of frames and the way they are decoded, the execution times for decoding the frames strongly depend on the type and the input data and vary extremely. An authentic WCET analysis as described in

section 2.2 is nearly impossible and missed deadlines cannot be excluded. The results can be realized in terms of incompletely decoded frames or unsteady movements in the video. Burchard *et. al*. [BUR 99] presented an alternative decoding approach using a predecoding phase as well as precalculated execution times per frame that would be stored inside the stream.

The deadlines for decoding the frames arise from the frame rate used for coding the video stream. If a deadline were missed, it is possible that the succeeding frame or multiple frames would have to be skipped. The reason is that in most cases a video stream is correlated with an audio stream, which is decoded separately. Hence, both streams must proceed synchronously and as a result some video frames have to be skipped if the video decoding is too slow. With respect to the frame skipping, the video decoding can also be regarded as a firm real-time system because late decoded frames are of no use.

1.1.4.1.3. Cell phone audio transmission

Digital telecommunication audio transmission between cell phones consists of many tasks that fulfill the mission of transmitting an audio stream from one human being to another one by a digital signal stream. Figure 1.2 shows an abstract and simplified presentation of the required transmission tasks.

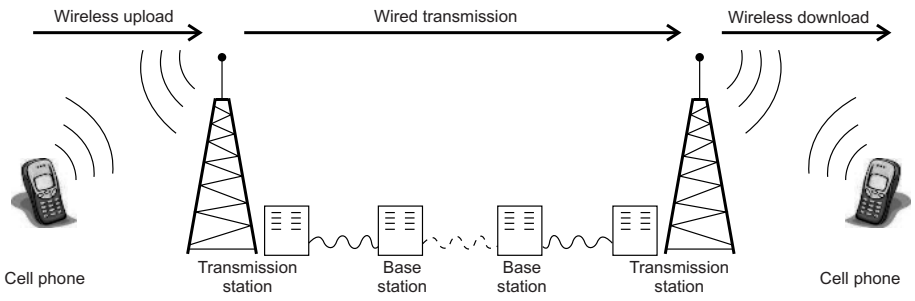


**Figure 1.2.** *Simplified task chain of an audio telecommunication transmission*

The five tasks, namely *sampling*, *wirelessupload*, *wiredtransmission*, *wirelessdownload* and *playback*, have to meet a single common real-time requirement. The transmission delay should not be noticeable by the participants. This requirement is not very strict because it is a subjective demand.

But, the challenge of telecommunication is to synchronize the required tasks in a way that all data reach the receiver in time. Hence, the sampling cell

phone has to meet deadlines to achieve a predefined sampling rate; the wireless upload as well as the wired transmission and the download must guarantee a certain data rate. The last element of the chain has to decode the received stream also using the predefined sampling rate.

The telecommunication sector itself represents a soft real-time system but because of the diversity of participating devices, companies and transport medias, it is a very complex area with respect to the real-time requirements.

#### 1.1.4.2. *Hard real-time systems*

The examples presented in section 1.1.4.1 illustrate some systems with soft real-time requirements. In contrast to these systems, the examples shown in this section can bring danger to human beings or machinery in cases of a missed deadline. Hence, these examples have to be classified as hard real-time systems.

##### 1.1.4.2.1. The air bag system

The air bag should reduce the risk of injury to passengers within a car in case of a crash. Therefore, the air bag control system evaluates the data received from multiple crash sensors. The front air bags must be released in the event of a front crash but they must not be triggered if the crash is from behind or from either side. In addition, at least two crash sensors must report a crash to eliminate fail functions caused by a defect of a crash sensor.

Different types of crash sensors are available: accelerometers detect a high acceleration, pressure sensors measure fast changes of the air pressure within the doors of a car and the so-called Crash Impact Sound Sensing (CISS) systems recognize the sound of deforming metal.

Signals of all these sensors are evaluated by an air bag control unit, which decides if an air bag must be triggered or not. Because the active period of an open air bag is only about 100 ms, the activation of it must be timed very well so that its efficiency is at the maximum. Besides the original velocity of the car, it is also important whether or not the passenger has used his/her seat belt at the time of the crash.

An additional timing constraint originates from other air bags: the opening of an air bag is about 160 decibels loud and brings the risk of acute hearing loss. To minimize this risk, the two front air bags must not be activated at the

same time and hence, the air bag of the co-driver is triggered slightly after the driver's air bag.

As real-time constraints of an air bag control system, a simple deadline is not sufficient. Moreover, an exact point of time is required at which the air bag must be triggered. In addition, calculating the point of time on the fly and in time leads to another hard deadline. Missing this deadline or activating the air bag at the wrong time could lead to inadmissible danger for the passengers. In this case, no air bag might be the better solution.

1.1.4.2.2. The antilock braking system

The aim of an ABS in cars and aircrafts is to take care of the grip during extreme braking. Therefore, it tries to avoid the locking of wheels and to sustain static friction, which must not change to dynamic friction. Otherwise, the adhesive force is reduced and the efficiency of the brakes is affected. In addition, preventing the wheels from locking keeps the steering active and the driver or pilot is enabled to control the car or aircraft during hard or emergency braking.

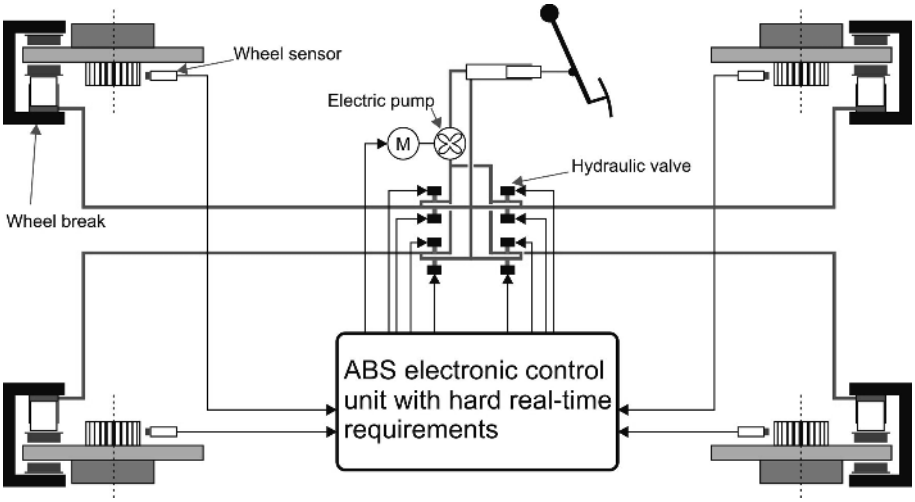


**Figure 1.3.** *Schematic of an ABS sensor and actuator system*

Figure 1.3 shows a schematic picture of the ABS of a passenger car. The control unit measures the rotation of each wheel independently, mostly by sensors using the so-called Hall effect [POP 04]. These sensors measure the

spinning of the wheel more than 50 times per rotation. If a wheel tends to get much slower than the others, the ABS control unit first opens an electromagnetic valve in the hydraulic braking system to reduce the pressure at the particular brake. In a second step, an additional pump is activated, which further reduces the pressure at the brake in order to release it. As a result, the wheel starts spinning again and the valve is closed once again to increase the compression.

Both procedures, opening the valve and closing it, have hard real-time constraints. If the control unit recognizes the locking of a wheel late, the braking effect of this wheel is reduced. Furthermore, if the system misses the time when the wheel properly spins, no braking is available because the valve is still open. Both scenarios could lead to horrible disasters and thus the ABS must be classified as a hard real-time system. Moreover, it is also a safety-critical system but these systems are out of the scope of this work.

1.1.4.2.3. Combustion engine control

Modern combustion engines are managed by a digital control unit, the so-called engine control unit. This unit is responsible for coordinating the ignition, the injection of fuel, the valve timing, the incoming air mass, the exhaust gas recirculation and so on. The main objective of using this electronic unit instead of mechanical motor management, as was used several years ago, is to reduce fuel consumption and environmental pollution. In contrast to the ABS and the air bag system, an engine control unit is not required to improve the safety of the passengers of the car. Nevertheless, if it is applied, it has to be classified as a hard real-time system, which is also safety related. The reason is, for example, that if the ignition is not correctly timed, the rotation speed of the crankshaft could become out of control and, in turn, the speed of the car would be out of control.

The mission of controlling the engine consists of several hard real-time tasks with different timing constraints. Some tasks such as the ignition control depends on the current angle of the crankshaft and some other tasks are related to timing periods. For example, the ignition must be activated several degrees before the corresponding piston reaches its upper dead center. Other tasks such as the exhaust gas recirculation must be managed periodically, independent of the crankshaft.

#### 1.1.4.2.4. Aircraft autopilot

Autopilots for aircraft are very complex systems, starting from multiple systems to control roll, pitch, yaw and throttle, and extend to a redundant system with, for example, three complete autopilots as required for airliners. The autopilot is an inherent part of an airliner and it is active during about 95% of a flight.

Besides the single stand-alone task of each autopilot subsystem, the tasks are related to each other by physicality. For example, during the flight the amount of fuel inside the tanks is reduced, which leads to a lower weight of the aircraft. Hence, to prevent the aircraft from climbing steadily, the pitch control reduces the pitch permanently to stay on a predefined flight level. Because of the lower pitch, the aerodynamic resistance decreases and the aircraft accelerates, which in turn increases the lift again. At this time, the throttle control has to reduce the speed to an optimal speed otherwise an overspeed situation can destroy the aircraft (see Figure 1.4).

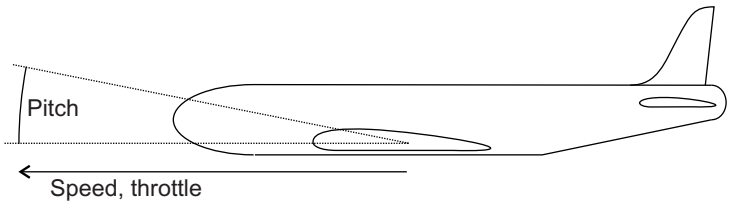


**Figure 1.4.** *Pitch and throttle of an aircraft*

In order to deal with these physical correlations, all the different tasks of an autopilot have to react to the current situation in time. In addition, the reactions of the tasks have to be coordinated and scheduled in order to prevent an oscillating situation where one task counteracts the actions of other tasks or the actions of two tasks accumulate. This could happen in the previous example if pitch control further reduces the pitch to prevent further climbing, while the throttle also decreases speed.

#### 1.1.4.2.5. Hydraulic motor control

An example without any direct impact on human life but on the functionality of a piece of technical machinery is the control logic of hydraulic motors. To increase the torque of the drill shown in Figure 1.5, two hydraulic motors are used in parallel. These two motors have to be activated

simultaneously to avoid damages. Because the motors are coupled physically, starting one motor before the other motor would force the second motor to rotate without oil flow. As a result, a low pressure inside it leads to a vacuum, which is explosively filled up by the following high pressure when the second motor is started. The problem is that this event could lead to a delamination of swarf, which can damage the hydraulic pump at a later point of time.

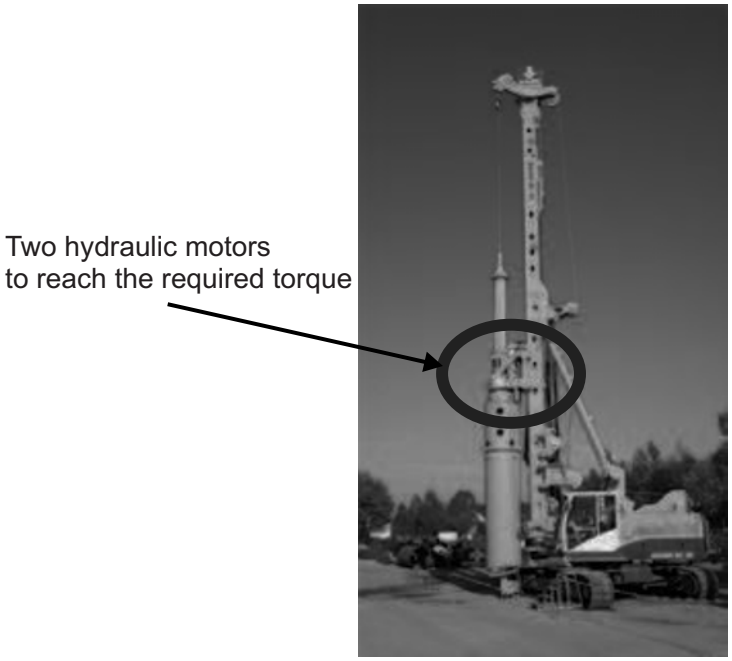


**Figure 1.5.** *Drilling machine with two hydraulic motors*

Of course, this seems to be a very simple real-time system compared to an autopilot because it is only required to open two valves simultaneously. But, if these valves are controlled by two different control units, a communication delay must be taken into account. In addition, this real-time scenario can be found inside an aircraft multiple times. Because of redundancy reasons, most rudders are moved by two hydraulic cylinders that are supplied by two hydraulic circuits and controlled by different control units. Hence, the problem of opening multiple valves simultaneously is also present as a small real-time task inside a very complex real-time environment.

## 1.2. Time predictability

Most of the computing systems exhibit high-performance requirements and the achieved performance of hardware and software is generally measured on the *average* case, i.e. on some executions that are considered as representative of most of the executions.

In hard real-time systems, aside from performance requirements, is the need for being able to show that real-time tasks can always meet their deadlines. For this purpose, the WCETs of critical tasks must be determined. This is possible only if the hardware architecture is time-predictable or time-analyzable. These two expressions are often used without distinction. However, Grund *et al*. [GRU 11b] formalize these concepts. They show that predictability (i.e. the ability to be predicted) is generally not a Boolean property, but can instead be expressed by a range of levels that allow comparing two systems ("A is more predictable than B"). Predictability can be seen as the ability to take into account a hardware scheme with a certain level of accuracy. The maximal accuracy is reached when the behavior of the system can be exactly predicted. Instead, analyzability is related to the approach used for timing analysis. It indicates the capacity of this approach to predict a given property.

A key point is that a real-time system does not necessarily have to exhibit high performance. Conversely, a high-performance processor may not fit the requirements of real-time systems. A common pitfall is to believe that a solution for making sure that a critical task will meet its deadline is to run it on a fast processor. Unfortunately, if the average execution time is usually improved by using a faster processor, this may not be the case for the WCET due to predictability/analyzability issues. The complexity of some of the schemes used to achieve high performance often makes the timing analysis complex and pessimistic. As a result, the WCET computed considering a high-performance processor may be longer than that obtained considering a simpler processor. In addition, the variability of execution times is generally higher when the processor implements sophisticated mechanisms and the observed execution time might be far from the estimated WCET, which may be a problem.

## 1.3. Book outline

The purpose of this book is threefold: (1) to offer an overview of the state-of-the-art techniques in the field of timing analysis of hard real-time systems, (2) to provide an insight into the difficulties raised by advanced architectural schemes with respect to such timing analysis and (3) to review existing techniques toward the design of time-predictable processors.

Chapter 2 provides background information on timing analysis approaches. First, an overview of task scheduling techniques, both for single-score and multicore systems, is given. These techniques all consider estimations of the WCET of tasks, i.e. their maximum possible execution time, whatever the input data values and the initial state of the hardware platform. Various approaches can be used to determine WCET estimates: some of them rely on measurements of the program execution on the target hardware, while other approaches consider models of the hardware as part of static program analyses. The main lines of such approaches are reviewed. The chapter ends with a discussion on the notion of time composability.

Chapter 3 focuses on current processor architectures. It studies the main schemes implemented in the execution core to enhance the average-case performance: pipelining, superscalar execution, multithreading and branch prediction. Each of them is first presented, and then state-of-the-art techniques to determine their worst-case timing behavior are introduced. Finally, directions to improve its timing predictability are discussed.

Chapter 4 deals with the memory hierarchy: instruction and data caches, scratchpad memories and the external main memory. The specificities of each level of the hierarchy are reviewed. Then, the basic existing approaches to analyze the worst-case behavior of such memories are presented and recommendations to build time-predictable memory systems are given.

Chapter 5 deals with multicore architectures that are now unavoidable when designing high-performance embedded systems. Multicore architectures are beneficial in terms of integration and power efficiency from the sharing of resources among cores. However, this sharing challenges the timing analysis of critical software because individual tasks can no longer be analyzed separately. At the same time, the global analysis of several concurrent tasks jointly does not seem feasible: first, this raises computational

complexity issues and second, the set of tasks that run concurrently is often decided dynamically. For these reasons, it is necessary to consider specific schemes to control resource sharing to some extent, in order to favor timing composability. This allows the use of timing analysis techniques that are close to those developed for single-core architectures. In this chapter, we review such hardware schemes.

Finally, Chapter 6 describes single-core and multicore architectures that have been designed in several academic and European projects, with time predictability as a main objective. These architectures may inspire future commercial designs once the view to time predictability spreads over application domains that require time-critical software design.