Chapter 2

Hard real-time computing environment

A man with a watch knows what time it is. A man with two watches is never sure. [Segal's Law]

Concepts introduced: Open and closed systems, RT entities, Clock accuracy.

THE ENVIRONMENT for hard real-time systems begins with prosaic concerns such as the details of the computer hardware that will be used. In practice we carefully choose appropriate components to improve the behaviour of the final system. We are concerned with the architecture of distributed systems of the computers, and with their synchronization, and notions of time and accuracy. The sum of this range of concerns leads to difficult problems; there are no perfect or best solutions, but we suggest various tools to apply to the problems.

The overall requirements for hard RT systems can be viewed in three areas:

1. **Functional:** What data must be recorded, what actions to take, the UI.
2. **Temporal:** How quickly the system reacts.
3. **Dependability, safety:** How reliable is the system.

Starting with the functional requirements, we may have to collect data/events (such as the position of a piston), condition these measurements (perhaps smoothing), and perhaps monitor alarms. The events which change with time are termed RT entities, and may be continuous, or discrete. An example of a continuous RT entity is the temperature, or the pressure in some container. A *discrete* RT entity can be observed only between specified occurrences of interesting events.

When our nodes measure RT entities, they may store its value and the time that it was recorded in the memory somewhere for later processing and analysis. This is called the RT *image*: $\langle \mathrm{Name}, \mathrm{Time}, \mathrm{Value} \rangle$. The accuracy of an RT image can be not only in the accuracy of the value recorded, but also, since it is only correct for a short time interval, in temporal terms:

Definition 5 *$\langle N, t, v \rangle$ is Δ-accurate if the value of N was v at some time in the interval $(t - \Delta, t)$.*

Suppose $\langle N, v \rangle$ is observed at time t and used at time t'. Then the maximum error $(v' - v)$ depends on the temporal accuracy (Δ) and the maximum *gradient* of N during this interval. If the gradient is high then Δ must be small and tasks using N must be scheduled often!

RT image	**Maximum change**	**V-accuracy**	**Δ-accuracy**
Piston Position	6000rpm	0.1degrees	3μsec
Accelerator pedal	100%/sec	1%	10msec
Engine load	50%/sec	1%	20msec
Oil temperature	10%/min	1%	6sec

Table 2.1: Accuracy of some RT entities

Consider also the actions the computer system must take: we have to design and implement control algorithms to calculate set points for the actuators, sample new values for the RT entities, calculate and output the new set points to the actuators, all the while taking into account delays and variations. A final important functional requirement is the man-machine interface. Many systems have failed at this point through poor design, but in this book we do not address this area.

Temporal requirements are paramount in a hard real-time system. The delay between a change in an RT entity and the actuator response should be minimized, although not at the expense of stability. In some systems an immediate response may result in oscillation, and it may be better to slowly respond as in Figure 2.1.

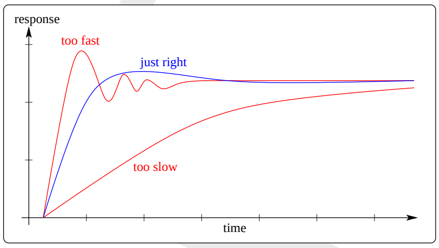

Figure 2.1: Optimal response to event

The study of closed loop systems and their response to impulses is outside the scope of this book, however we do investigate simple temporal properties:

Definition 6 ***Dead time*** *is the delay between the observation of an RT entity and the start of the reaction at the plant. This delay may be the sum of a delay from the computer, and a delay inherent in the plant itself.*

In addition to minimizing dead time, it is also likely that we would need to minimise the latency jitter, which is the difference between the maximum and minumum delay from the computer.

The final requirement view was with dependability and safety. Since the application areas of hard real-time systems tend to be in highly critical areas (cars, plane flight systems, nuclear reactors, weapons, spacecraft), we require the highest level of reliability. If the failure rate of a system was $\lambda \mathrm{failures/hr}$, then the MTTF (Mean Time To Failure) is $\frac{1}{\lambda}$ hours. A failure rate of $10^{-9}\mathrm{failures/hr}$ may be considered good enough for a car.

2.1 Hard RT controller systems - architecture

Monolithic system architectures are discouraged in modern RT systems, particularly as a failure in any one area should not affect other parts of the system. We would not want the brake system of a car to fail at exactly the same time as the steering system, but that is what would happen if a single (monolothic) computer system was responsible for all such activity in a car.

Instead it is common to construct hard RT systems from a set of distributed computers (in the abstract - *nodes*).

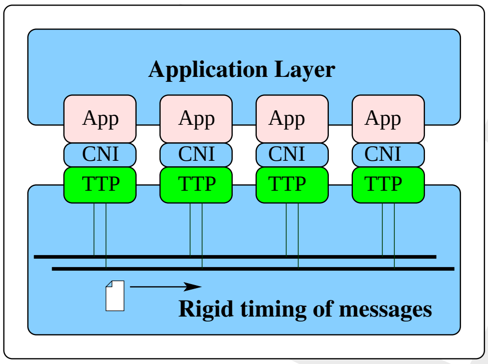

Figure 2.2: Timing of the communication infrastructure

For RT systems, the underlying communication infrastructure is rigidly timed as seen in Figure 2.2. This means that we can guarantee the timing of the delivery of messages from node to node. In many other communication systems (for example ethernet's CSMA/CD), the underlying communication infrastructure relies on probabilistic access to a shared (wire) medium, and we cannot guarantee delivery. It would be unfortunate if the brake signals on a car were delayed, because the shared communication medium was busy reporting the temperature of the engine.

The nodes connect to each other via two independent channels. The communication subsystem executes a periodic Time Division Multiple Access (TDMA) schedule, reading a data frame and state information from the CNI (Communication Node Interface) at predetermined times, and delivering relevant information to the CNIs of all receiving nodes at predetermined (scheduled) times. All the TTPs in a cluster know the schedules, and so it is important that all nodes of a cluster have the same notion of global time. Each clock must have some fault-tolerant clock synchronization system to ensure this.

This general architecture is also interesting because of the possibility of replicating/duplicating nodes for the purposes of fault tolerance.

Each node consists of a host computer performing some function (or set of functions) with access to the communication infrastructure through the CNI (a networking API). Once configured, the CNI provides guaranteed-time delivery of data if needed. The host computers may have a wide range of capabilities, but will always include some I/O, and a local clock. Each processor may have internally some set of tasks, each reacting to some external event of interest. In the hard RT environment, we are principally interested in those events which change continuously with time.

Note that we have started to identify the *core* elements of RT systems; those that separate this study from other system software engineering paradigms. In particular, we focus on temporal properties.

2.2 Hard RT controller systems - hardware

FlexRay is a fault-tolerant, deterministic time-triggered communications protocol, developed since 1999 by automobile manufacturers. The initial developers were BMW and Daimler/Chrysler, who recognized that the existing event-triggered systems (principally the CAN bus, but also ByteFlight, J1850, LIN, MOST) were not safe enough, and could not form the basis for hard real-time systems needed for modern car automotive controls.

Chip and component manufacturers (such as Freescale) have adopted the technology, and specifications, software and hardware for FlexRay is available. Cars have been released using FlexRay, and most car manufacturer's prototypes are using the technology. It is expected to be the standard in 2 or 3 years.

FlexRay is not only oriented towards cars, but also avionics and space systems. It includes both synchronous and asynchronous modes of transmission, and is scalable. The initial target has a $10\mathrm{mbps}$ gross data rate, but the *protocols* support much higher data rates. The hardware (chips) available today provide a $5\mathrm{mbps}$ net data rate, and using the synchronous mode, the hardware directly supports time-triggered communication. By contrast, the CAN bus maximum data rate is $1\mathrm{mbps}$.

An example development kit for FlexRay is the one from Fujitsu, which consists of an FPGA board with the FlexRay communication controller, the physical layer, and an MCU starter kit equipped with the MB91369, a 32-bit FR50 core. The 32-bit RISC core features a Harvard architecture, with a five-stage pipeline offering a single cycle instruction execution.

The MCUs have 512kByte of embedded flash memory as well as 24kByte of embedded SRAM and an external interface for direct access to 16-bit SRAM as well as asynchronous ROM. There is a high-speed 5-channel Direct Memory Access Controller (DMAC), serial communication with an I^2C interface and up to five USARTs.

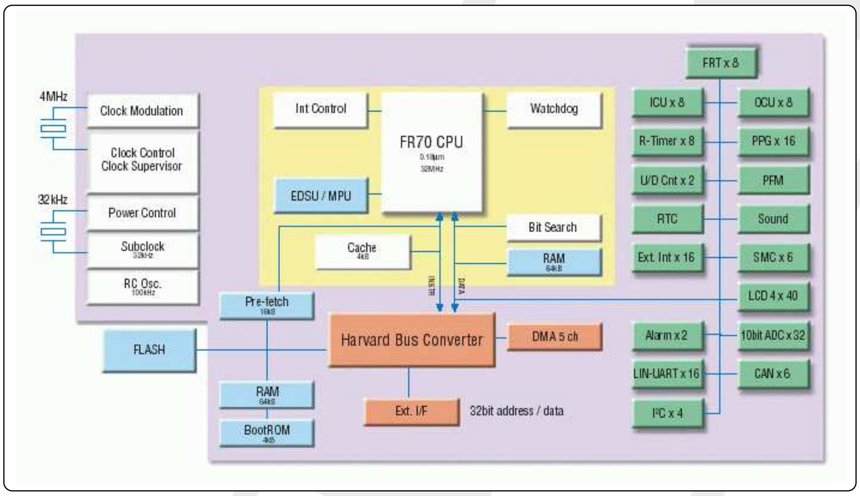

Figure 2.3: FR50 MCU architecture

Four independent ICU (Input Capture Unit) channels are included together with up to eight OCU (Output Compare Unit) channels, four 16-bit reload timers and up to six PPG timers. Up to 12 channels of 10-bit A/D converter and up to three channels of D/A converter are available. It is becoming common now for in-car electronics to have this level of processing and I/O power.

Along with the hardware comes a software package featuring a ready-to-use communication example. The software package includes a communication stack, which meets series production requirements for evaluation purposes or prototype development of FlexRay products. It is common now for in-car electronics to have this level of processing power.

2.3 Clocks and synchronization

When constructing a single system from a set of cooperating nodes, we have to ensure that each node has the same view of time. The technique used is for each node to maintain its own clock/time, so that in the event that the system loses contact with (perhaps) some global system clock, it can continue working on its own until it re-establishes a link to the global system clock.

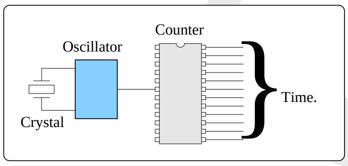

Figure 2.4: Clocks in computers

Clocks in computer systems are implemented using counters as in Figure 2.4. The counter counts up at a precise (*microtick*) rate determined by a vibrating (piezo-electric) crystal. These crystals vibrate at a fixed frequency, typically 32KHz, or 1 MHz, or 16MHz. They are not perfect, depending on the temperature and voltages supplied by the oscillator, and they will drift slowly away from a precise reference source by an unbounded amount. Typical drift rates are 10^{-2} to 10^{-7} secs/sec.

Figure 2.5: Patriot missile system

On Feb. 25, 1991, on a patriot missile defense system, the accumulated drift over a 100 hour continuous operation (never before experienced) was nearly 343 msecs. This led to a tracking error of 687 meters causing an incoming Scud missile to be declared a false alarm. As a result of this, many people died or were injured.

In order to avoid such things we need to be precise about clocks and their behaviour. Our system view of time is based on this hardware-oriented view of clocks as counters of *microticks*. Since we may have multiple clocks in our system, we imagine a reference (perfect) clock ticking away, and reason about our clocks *relative* to it.

Definition 7 *The term **drift** refers to the frequency ratio between a clock and a reference clock (over a particular time segment). The perfect value is 1.*

Definition 8 *The term **offset** refers to the time difference between the microticks of two clocks measured in terms of the microticks of the reference clock.*

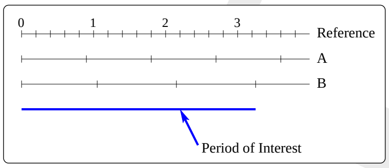

Figure 2.6: Example of clock drift

We can illustrate the terms precision and accuracy by considering the simple example in Figure 2.6 where 5 microticks of the reference clock correspond to 1 microtick of two clocks A and B. The period of interest is the first 3 microticks of A and B.

Definition 9 *The term **precision** refers to the maximum offset found between a set of clocks measured in terms of the microticks of a reference clock.*

In Figure 2.6, the maximum offset occurs at tick 3, where the precision is 3 microticks.

Definition 10 *The term **accuracy** refers to the maximum offset between a clock and the reference over a particular period of interest.*

Clock A is running fast, and clock B is running slow. Over the period of interest, the maximum offset of A occurs at A's tick 3, which occurs at 13 microticks of the reference clock (an offset of 2 microticks). The maximum offset of B occurs at B's tick 3, which occurs at 16 microticks of the reference clock (an offset of 1 microticks). The accuracy of the collection with respect to the reference is 2 microticks.

Definition 11 *The term **granularity** refers to the time between two ticks of the clock.*

Assume we have a collection of two clocks A and B whose precision is 10 msecs and whose global time granularity is 8 msecs. It is possible for B to report that events e and e' occur at the same time, although A reports that the events occured one after the other (see Figure 2.7).

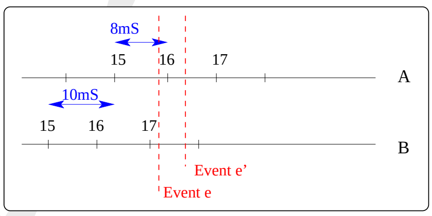

Figure 2.7: Event ordering with two clocks

Without this analysis, we may have two systems drifting apart in time, and incorrectly measuring the temporal or causal ordering of events.

What are temporal and causal orderings? Temporal order refers to the order in time at which events occur. Time is modelled by using a totally ordered set of instants, and events occur at instants. The ordering of the instants associated with particular events indicate the temporal ordering of the events. Causal order refers to an ordering between events related by causation; that is if `event1` occurs, then `event2` must follow, as `event1` causes `event2`. Since we cannot effect things in the past, a causal order must imply the same temporal order; that is if `event1` causes `event2`, then `event2` must occur after `event1`.

It is necessary for the clocks in our system to maintain internal or external synchronization. Internal synchronization refers to ensuring that clocks in a connected system maintain a bounded precision (they are all within x msecs of each other). External synchronization refers to ensuring that the clocks maintain a bounded precision with an external reference (they are all within y msecs of UTC).

Coordinated Universal Time or UTC, Zulu time or Z, is a related (atomic) realization of GMT (Greenwich Mean Time), the basis for civil time. Time zones around the world are expressed as positive and negative offsets from GMT. UTC differs by an integral number of seconds from International Atomic Time (TAI), as measured by atomic clocks and a fractional number of seconds from UT. From time to time, leap seconds have to be inserted into this time scale.

TAI is a very accurate and stable time scale. It is a weighted average of the time kept by about 300 atomic clocks in over 50 national laboratories worldwide. In these clocks, one second is the duration of $9,192,631,770$ periods of radiation of a specified transition of the cæsium-133 atom, intended to agree with the time derived from astronomical observations. In this time standard, there are no *leap* seconds.

It is common to synchronise the clocks of computer systems using the Network Time Protocol (NTP), a protocol for time synchronisation over packet-switched, variable-latency data networks (such as the Internet). NTP uses Marzullo's algorithm with the UTC time scale, and can maintain time to within 10 msecs over the Internet, and 200 μsecs or better in local area networks. NTP uses a hierarchy of clocks, where the highest level clocks are synchronised to an accurate external (atomic) clock. Level 2 systems derive their time from one or more level 1 systems, and so on.

Marzullo's algorithm finds an optimal interval from a set of time estimates. The best estimate is taken to be the smallest interval consistent with the largest number of sources. If we had the intervals $[8, 12]$, $[11, 13]$ and $[10, 12]$, the intersection is $[11, 12]$ or 11.5 ± 0.5. Rather than take the center of the interval as the value, as specified in the original Marzullo algorithm, NTP also uses other useful information about the sources to return a more refined value.

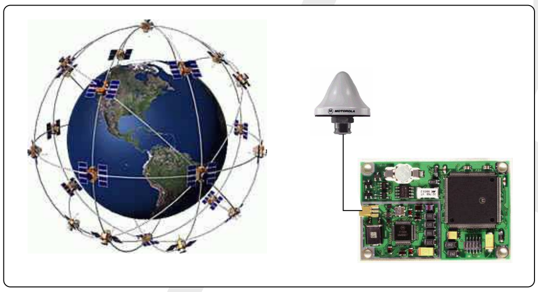

Figure 2.8: M12+ timing receiver for GPS time

If NTP is not accurate enough for a particular RT system, nearly 100 atomic clocks orbit the Earth in the Global Positioning System (GPS). At any time any point on the globe is within range of at least four satellites, and receivers such as the Motorola M12+ timing receiver can establish a precise time to within a few nanoseconds.

2.4 The design challenge

The grand challenge for hard real-time systems is to derive a model of the closed system, and then design and implement a distributed, fault-tolerant, optimal, real-time computing system so that the closed system meets the specification/requirements.

Each computing node will be assigned a set of *tasks* to perform the intended functions. These tasks involve the execution of a (simple) sequential program which reads the input data, and using the internal state of the task correctly processes it, before terminating with production of the results and updating the internal state of the task. Most tasks will have some internal state; if not they are termed stateless.

The real-time operating system provides a control signal for each initiation of a task. Simple tasks are ones that have no synchronization point within the task. They will not block due to lack of progress by other tasks in the system, but they can be interrupted (preempted) by the operating system. The total execution time for these tasks can be computed in isolation: the WCET (Worst Case Execution Time) of the task over all possible relevant inputs. A correct estimate of WCET is crucial for guaranteeing real time constraints will be met.

More complex tasks contain blocking synchronization statements. For example the `wait()` semaphore operation, or the `receive()` message operation. The WCET of a complex task can not be computed in isolation.

Tasks may be triggered by exceptions, interrupts and alarms. There will be tasks that need to be executed periodically, and these tasks may have precedence relationships, deadlines, and share data structures. They may also have to execute on the same processor. We must schedule!

2.5 Summary of topics

In this section, we introduced the following topics:

Requirements for hard-RT. *Functional, temporal and dependable.*
Architecture. *We outline the time-triggered architecture, with an example (FlexRay).*
Clocks. *Timing and synchronization study, with definitions of the important terms.*
A challenge. *Design distributed, fault-tolerant, optimal, real-time computing systems*

2.6 Supplemental material

2.6.1 Exercises for Chapter 2

1. Develop a timed-automata diagram for the train.
2. Develop a timed-automata diagram for the gate controller.

2.6.2 Recommended reading

❖ Time-Triggered Paradigm [EBK]
http://citeseer.ist.psu.edu/elmenreich03timetriggered.html