

**GEINTEGREERDE SYSTEMEN
VOOR DIGITALE
SIGNAALVERWERKING:
ONTWERPCONCEPTEN EN
ARCHITECTUURCOMPONENTEN**

**INTEGRATED SYSTEMS FOR REAL-
TIME DIGITAL SIGNAL
PROCESSING:
DESIGN CONCEPTS AND
ARCHITECTURE COMPONENTS**

ACADEMIEJAAR 98-99

**F. CATHOOR - H. DE MAN
K.U. LEUVEN
ESAT-INSYS**

CHAPTER 2: SPECIFICATION AND ALGORITHMIC REFINEMENT OF REAL-TIME SIGNAL PROCESSING

3.1. CHARACTERISTICS OF REAL-TIME MULTI-MEDIA AND TELECOM SIGNAL PROCESSING

Many industrial systems treat data information in synchrony with some "periodic source" or "stream", controlled by a single sample period T_s or a set of periods which are multiples of a "basic" period (Fig.2.1). This is called "real-time" or "on-line" processing. As illustrated in Fig.2.2, the sample rates F_s at which the data source is producing the input stream, typically exhibits a very wide range. In particular, this goes from a few Hz (in control systems needed in automotive applications or measurement as in medical diagnostics) over tens of kHz (receiver-end telecom, audio, speech, compact disk) to tens or hundreds of MHz (imaging, video, front-end telecom and radar). It has to be stressed that this sample rate requirement is a behavioural specification issue. This is quite different from a clock rate F_{cl} that is related to the final realisation! Note also that the concept of sample can have many interpretations. For instance, in video it can be frames, lines, pixels, or even combinations of these, dependent on the subsystem that is being considered

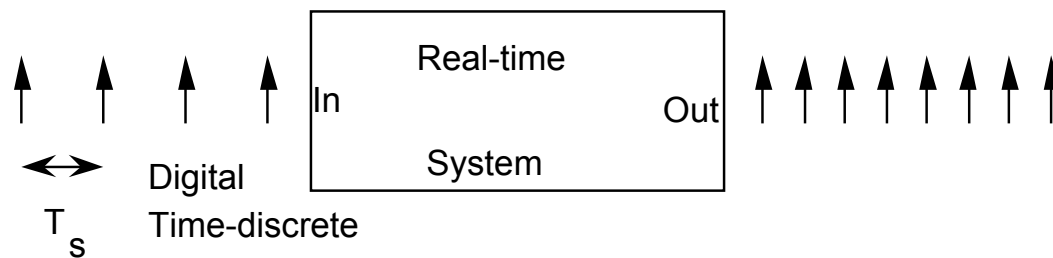


Fig.2.1 Real-time system

F_s	Hz	KHz	MHz	>100 MHz		
	Control	Audio	Front-end	Medium	Front-end	
	Measur.	Speech	Audio	Level	Image	Radar
		Telecom		Image	Video	Telecom
		(User-end)		Video		
		Real-time Anal.		Radar		
		Back-end Imag.				

Fig.2.2 Range of real-time signal processing application domains

The throughput at which samples are treated by the system can be fixed or (slightly) varying. As long as the data source is providing a continuous stream of samples however, the system is considered to be working in real-time. This basic characteristic is providing the distinction with off-line task-based systems where the input happens once and where a particular "task" has to be executed, usually in a specified response time. Typical examples of applications are numerical computing where the input

occurs e.g. via a terminal and where the data to be treated is first stored on a (large) disk.

In this course, only digital and mainly synchronous systems will be considered where a realisation is governed by a master clock. In such digital systems the "signals" can be of several data types (see also CAD part and manual of LOGMOS2): scalar (Boolean 0-1 or bit 0-1-x), word (ordered array [1..n] of bit), vector (array [1..k] of word), matrix or block (array [1..k,1..l] of word). Multi-dimensional signals are very common in real-life signal processing applications. This complicates the design process for industrial systems considerably. Moreover, a certain information content or meaning is attached such as integer, ASCII bytes, twos complement data, 8-bit intensity of an image pixel, 16-bit compact disk sample and so on.

The data-streams at the input have to be processed by means of an "algorithm" in order to be transformed into the desired results. This can take place under influence of or steered by the "external" world. Moreover, flags or status signals (e.g. a sign bit of an intermediate result) can be produced in the course of these algorithms which also allows to influence the algorithmic "behaviour" by means of conditions (IF-THEN-ELSE, see further on).

Therefore, in most of these signal processing applications, computation-intensive arithmetic operations have to be combined with complex decision-making (a nesting of conditions and loops). Specification issues for this functional behaviour will be treated in more detail in the CAD part. Usually, a textual representation in a procedural or an applicative language is more amenable for DSP systems but designers also like graphical representations, especially for the top level of the system hierarchy.

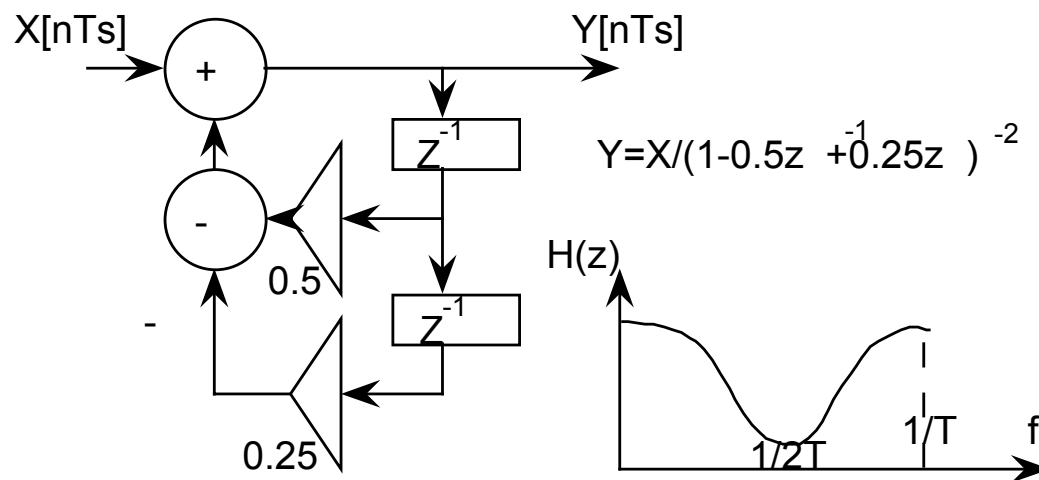


Fig.2.3 2nd order digital IIR filter: SFG (a) and transfer function (b)

In order to keep this discussion less abstract, we shall use an all-pole 2nd order digital filter as a test-vehicle to demonstrate the main differences between the major architectural styles in Chapter 3. This is a simple DSP subsystem that is presented in Fig.2.3a. It is a typical example of a data independent, time-invariant signal processing algorithm which operates in real-time on a sampled data stream $x[n.T_s]$, where $n=0..1$ (discrete time index) and T_s is the sample period (see course on discrete-time systems

of Prof. Vandewalle). The transfer function of this test-case is shown in Fig.2.3b. Can you formulate a closed formula and retrieve this response? In most cases, such an algorithm is represented by a data dependency graph (DDG) where the nodes are data operations (such as add or compare) including data storage/retrieval and where the directed arcs represent the dependencies of inputs to the sink nodes from the outputs of the source nodes. In real-time signal processing however, the input and output signals are assumed to be continuous streams (in principle of infinite length) so it is impossible to represent all instances of the time axis in a single (acyclic) DDG. Therefore, the most suited representation considers only 1 time slot (one sample period T_s) on the time axis with all the signal instances produced/consumed during this slot. Such a representation is called a signal flow graph (SFG). Fig.2.3a is an example of such an SFG. Signal instances from previous time slots can be referred to as “delayed signals”, represented with the z^{-1} operation in the graph.

3.2. DATA TYPE REFINEMENT

As already mentioned in the introductory chapter, the original data types (sometimes called “abstract”) are typically not yet suited to be mapped directly on the final realisation hardware. In order to overcome this problem, they have to be refined into “concrete” data types, matched to a minimum cost realisation target. The target of this stage are mainly the structured data types which have a large impact on the final realisation cost, but also scalar data types should be assigned their final type selection. The decision for structured data types has to be performed before the corresponding DTSE stage is entered. In practice, this typically means it is decided just after algorithm optimisation.

In most cases however, the decision for scalar data types can be postponed till just before the custom processor synthesis or instruction-set processor mapping stages in the system design trajectory.

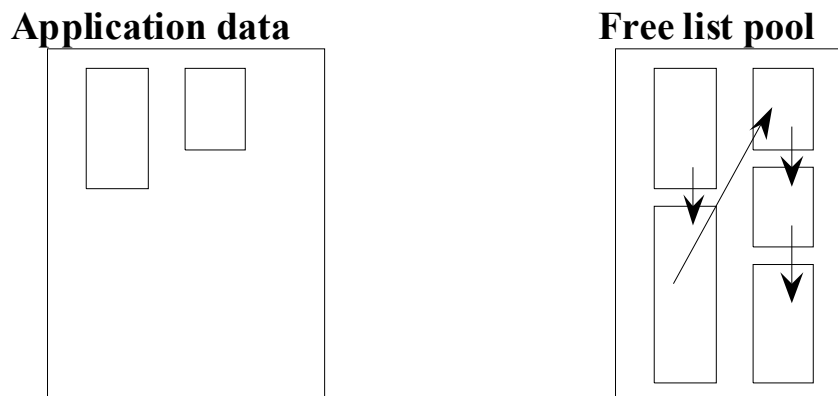


Fig.2.3 Illustration of virtual memory management for dynamically allocated data

This stage involves three steps, which should normally also be applied in the given order:

1. parameter quantisation:
 - selection of detailed data type for parameters in the algorithm.
 - Examples include digital filter coefficient optimisation, modulation scheme optimisation in terms of coding parameters, arithmetic coder table optimisation.
2. dynamic data type refinement:
 - selection of detailed data structure realisation of dynamic data types in the

algorithm.

Examples include refinement of dynamically allocated ordered set with access keys into as linked list, binary trees or pointer arrays; or virtual memory management for dynamically allocated data (Fig.2.4.5).

3. data format selection:

selection of detailed realisation of static data format for all variables in the algorithm. This involves both floating-point versus fixed-point and finite word-length decisions.

Finite word-length issues have been studied especially for linear DSP applications based on quantisation, limit-cycle, overflow and dynamic range requirements. Recent research is however also focusing on more general application.

3.3. TASK-LEVEL SYSTEM ARCHITECTURE DESIGN.

At the task abstraction level, both data transfer and storage exploration and concurrency management issues can be decided already before the detailed instruction/operation view is finalized at the processor abstraction level. DTSE issues will not be considered here because this is still a research issue. Similar decisions as for the processor-level in Chapter 6, Section 7 are relevant here however. The other issues are grouped in the task concurrency management (TCM) stage.

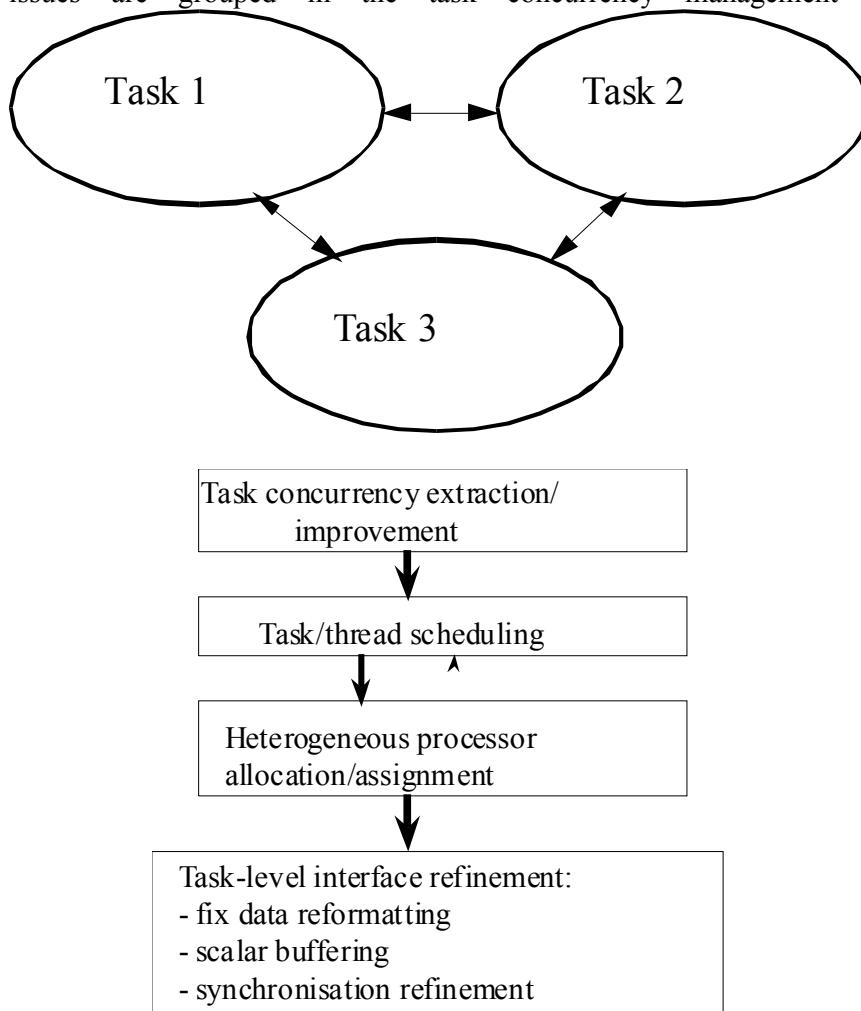


Fig.2.5 co-operating, communicating tasks: concurrency issues

Definition of TCM:

Stage in the system design trajectory where the task-level concurrency issues are decided on co-operating communicating system tasks.

The following steps are typically important (see Fig.2.4.5):

1. task concurrency extraction:
The first thing that has to be done is the analysis of the opportunities for task-level parallelism in the system. The limit imposed by inter-task data dependencies are examined in this part of the meta flow. Also transformations can be applied to arrive at a better "task" partitioning.
2. task/thread scheduling:
The various tasks are (partially) ordered and a relative time is decided for their interaction. In case a task with multiple threads of control is assigned to one heterogeneous processor, an additional scheduling step takes place to decide the start time of each thread in the task.
3. heterogeneous-processor allocation:
The designer makes a decision on the number of heterogeneous processors that will be used. A balanced choice driven by real-time constraints and maximum available parallelism should lead to a minimal cost allocation.
4. task to heterogeneous-processor assignment:
Here, tasks are assigned to heterogeneous processors. The threads contained in a task are shared on the heterogeneous processor it is assigned to.
5. inter-task interface refinement:
The interfacing requirements of communicating tasks are gradually refined in terms of data type conversion, scalar buffering including foreground storage units and interconnect, and synchronisation refinement.

Note that most of these steps have strong dependencies in the "direction" of the given order, but the relation between "ordering" and "processor allocation/assignment" is more complex because the final ordering decision may depend quite heavily on the processor a set of tasks are assigned to.

Still, in the majority of the proposed approaches and contexts, the ordering has the highest priority and that is decided first. If necessary, one or more local iterations need to be executed between these two steps. An alternative which some researchers prefer is to perform both steps simultaneously but the complexity of the combination is then very large.

Brain-teasers and examples:

- 1) an algorithm is implementable in a digital system only if the SFG is free of delay-free loops (i.e. without data storage). Why?
- 2) a realizable SFG can be represented and simulated in a functional language with single assignment, without making a hardware choice as yet. Why? Based on this principle, efficient system description languages have been proposed, such as SILAGE which is used as the input for the CATHEDRAL-II (developed at IMEC) and HYPER (developed at U.C.Berkeley) architecture synthesis environments.