7.

Conclusions and Future Work

This chapter presents the general conclusions about the work presented in this dissertation. Besides observations about the methodology as a whole, extracts from the summaries shown at the end of each previous chapter are also discussed. At the end, directions towards future work are also commented.

7.1 Objectives

It is important to stress again the primary objective of this dissertation. The majority of the co-design approaches developed so far have not stressed issues related to real-time systems. Partitioning techniques, hardware-software interfacing and co-synthesis, and software speed-up have been the major topics in the area. Therefore, a primary objective was set to work in this niche and develop a system able to, from an initial functional description, produce architectures composed of several software hosts working in a pre-emptive fashion and co-operating with hardware components able to operate in real parallelism, and yet fulfilling critical temporal requirements.

7.2 Summary of The Dissertation

The state-of-the art on hardware-software co-design was described in Chapter 2 together with an overview of the subject. Several sub-divisions of co-design were presented, namely co-specification, partitioning, co-synthesis, analysis, as well as co-simulation and prototyping.

Although the interest in the development of micro-controlled embedded systems, and as such co-design, is not new, a great deal of attention has been noticed since the beginning of this decade. This reflects the need in the development of formal techniques for dealing with the boom in the use of embedded systems, bringing more complexity to the area but also requiring low cost, short design time, and flexibility (upgradability, maintainability, etc.).

The development of high-level synthesis, ASICs and FPGAs, on the hardware side, played a major role in the progress of the area since those techniques and devices made it possible to treat hardware in a design level similar to what was already common practice in software development.

Chapter 2 also gave a brief introduction to real-time systems and Petri nets, necessary to the proper understanding of this dissertation.

Chapter 3 exposed an overview of the co-design approach developed at the University of Newcastle and described in this dissertation. The system is targeted towards critical real-time embedded designs and consists of three major sub-parts:

- Specification tool (Time Wizard)
- Pre-runtime-based scheduler
- Hardware-software partitioner

Chapters 4 and 5 presented the scheduling and partitioning techniques, respectively. The scheduler is based on a pre-runtime technique able to deal with multi-processor architectures with hard real-time constraints, in the presence of precedence, exclusion, and pre-emption relations.

The partitioning method is divided in two phases: Pre-partitioning and System Partitioning. The former deals with contention to the use of the processor caused by task inter-dependencies, and exploits execution speed-up as a solution. The latter deals with contentions related to high processor utilisation, and uses parallelism as primary agent in the search for feasibility.

Chapter 6 has shown some experiments in the environment use. A mixture of real-life and academic examples were utilised in order to show different system aspects. The sections below draw some conclusions about each part separately and their iteration with the others, followed by plans for future work.

7.2.1 Specification Method

The design starts in a reasonably loose fashion and, based on information given by the analysis tools, refinements are carried out in order to achieve a consistent model to be used by the scheduler and the partitioner. The level at which this model is produced is called *scheduling level*. Specification is based on High-Level Time Petri Nets, with some extensions added to an already existing formalism. The method has been shown to be quite simple and yet powerful enough to provide design assessment at both internal and task interconnection levels.

Although Petri-net based specifications are not new in co-design, their use during all design stages is. Normally Petri nets are used as internal representation (e.g. [38]) or for analysis purposes only (e.g. [12]). This uniformity promotes a better understanding and control of the design process by the user, since there are no awkward internal representation methods and there is no need to master several different modelling paradigms. Apart from being used for general system analysis and specification, Petri nets have also been used in hardware high-level synthesis [3, 15, 121]. This would make it possible to extend the methodology presented here towards comprising all abstraction, levels from the initial specification towards the final architecture implementation. It is possible to translate specifications written using other languages to the formalism employed here.

The extensions proposed are aimed at increasing the usability of Petri Net-based design and real-time design analysis. It also promotes the development of automatic tools towards co-design and scheduling, such as the generation of the scheduling model presented in Time Wizard. Hierarchical modelling makes it easier to handle complexity through abstraction of low level details. The restrictions imposed in the hierarchical constructors guarantee the maintenance of general Petri net analysis properties.

The specification method used in this dissertation, although meant to be used in conjunction with the other co-design tools developed here, can be used separately as an aid for real-time design in general. In particular, the scheduling level output format generated is suitable to be used by all pre-runtime schedulers which I came across.

Far from being a problem only found here, the approaches used in pre-runtime scheduling design proved different from common software development techniques. As such, they are difficult to get used to mainly because here tasks are grouped based on their triggering event. Software engineering techniques, on the other hand, reward grouping tasks which are functionally similar or that perform actions over the same type of object, e.g. in object-orientation languages. This is already been done for real-time systems, such as in HRT-HOOD [21], but only when using fixed-priority dynamic scheduling policies. Perhaps, as spotted by Kopetz [74], the best solution would be to join both techniques in order to keep the advantages of each one.

Time Wizard, the CAD tool which incorporates this specification methodology, is implemented in 10,790 lines of code using Microsoft Visual C++® v1.52 for Windows 3.1X®.

7.2.2 Scheduling Algorithm

A branch-and-bound algorithm is used as the basis for scheduling. At each node in the search tree an attempt is made to produce a feasible solution. New nodes are created where changes (tighter inter-task relations) are incorporated on predecessor nodes data so that new possibilities are tried. Pruning techniques are employed to promote the early removal of paths which have no chance of producing feasible solutions.

The branch-and-bound approach, contrary to heuristic approaches, is able to find a solution whenever one exists. Similar approaches have been used with success in large real-time systems, such as flight and weaponry (mission) control systems for the F-18 fighter [108].

To the best of the author's knowledge, there is no other algorithm able to produce a solution whenever one exists for the pre-runtime scheduling of a set of processes with deadlines, release times, precedence, exclusion relations, and which allows process pre-emption, in a heterogeneous multi-processing architecture. Also, this is the first time that a pre-runtime scheduling algorithm of any sort has been used in co-design.

The capabilities of scheduling sets of previously allocated tasks on multi-processor architectures makes it very attractive for a diverse range of problems, from task to operation-level scheduling, from prototyping (with the use of several software hosts) to the final implementation. A uni-processor architecture may prevent the full exploitation of parallelism among tasks, and may overflow the processor utilisation capacity which causes more operations to be moved to hardware. In designs where the cost of hardware is the main concern, this may be critical. Further, all-software solutions (where no ASICs are used) may become unfeasible. These sort of solutions are very important, e.g., for rapid design prototyping.

The scheduling algorithm is completely separated from the partitioner although the contrary is not true. It can operate independently from the rest of the system and so can be used in other real-time design systems that do not feature co-design partitioning. In fact, its implementation comprises independent input methods to make it able to be used without Time Wizard.

Co-design particularly benefits pre-runtime approaches by coping with problems related to highly responsive sporadic tasks. Such tasks can be moved to hardware and dealt with properly. Transformation techniques can be used so that the bit that requires high responsiveness may be separated from the rest of the task. Since this part is relatively small, the onus in moving such a task to hardware would not be big.

At the same time, static schedulers do not require complex operating systems in the design implementation and context switch can be made short since the exact points of their occurrence in the processes are known in advance. Thus, this is a very attractive approach for embedded devices since computational resources are normally scarce.

The current implementation is computationally intensive, requiring large amounts of memory. As an illustration, the York2 example presented in Chapter 6 required 32MB of RAM memory in order to execute. However, this is not an optimised implementation since many intermediate data is being kept to verify the algorithm itself. Also, since the actual scheduling is calculated prior to the design execution, the time spent on the production of the solution does not affect the tasks execution times. At the same time, the branch-and-bound algorithm could be made faster by simply spreading it over several processors using a processor farm architecture, i.e., a master sends commands to slave processors requiring some data calculation, in the case here the production of a node scheduling [63]. The scheduler is currently implemented in the same software program as the partitioner. Both together are implemented in 14,540 lines of code using Microsoft Visual C++® v1.52 for Windows 3.1X®.

On the down side the scheduling technique shown offers little flexibility for design changes and upgradability after design implementation. These characteristics, together with the handling of sporadic tasks, are better treated on fixed-priority dynamic schedulers. However, these methods are not good when it comes to systems presenting task inter-dependencies, which is a feature of pre-runtime schedulers. Again, the solution may be to join both scheduling techniques.

7.2.3 Partitioning Algorithm

The partitioning algorithm is aimed at finding feasible solutions with regard to real-time constraints, and as such has temporal requirements as its primary concern. It is divided into two parts:

- Pre-partitioning - Tasks are analysed regarding their own constraints and their relations with other tasks. Solutions to this phase require the speed-up of task execution times, by moving those tasks to hardware or to a faster software host.
- System partitioning - No particular inter-relation is taken into account except for contention among tasks towards processor utilisation. The result of this phase may be found only by parallelism exploitation, and so possible solutions could be found by moving tasks to other processors, even with the same processing power, but which are less utilised. However, the approach used here attempts only to move tasks to hardware.

The partitioning method, with its separation in two phases, and incorporation of scheduling techniques for real-time properties verification and data generation for automatic or assisted partitioning, is original in the co-design field.

The method of selecting candidates to move to hardware found in the pre-partitioning algorithm can be used in any sort of real-time system, either using pre-runtime techniques or not. At the same time, this two-phase separation between solutions that require execution speed-up and those which can be solved through parallelism, seems a promising approach for other real-time partitioning and even allocation systems. In particular, the incorporation of the techniques developed here for use with rate-monotonic and other fixed priority real-time systems is left as future work.

The partitioning parameters used are similar to those found in other works, such as [40]. However they rely more on precise timing data generated by the scheduler, whereas the normal approach is to perform partitioning at higher abstraction levels. This is explained by the need to gather more data when it comes to guarantee time critical systems. Although pre-partitioning uses only data which regards static inter-dependencies, system partitioning has to do with the dynamic interaction between tasks when executing. Therefore, the partitioner works closely with the scheduling algorithm shown before in order to obtain the required information about the dynamic system behaviour.

Not only automatic but also user-assisted partitioning is present in the algorithm implementation. Whenever partitioning lists or sets are generated, their cost parameters are calculated and presented to the user, together with a mark on the task chosen by the partitioning algorithm to be moved to hardware. Then, the designer can accept the hint or make his own choice of task to move.

7.3 Future Work

This dissertation provides a starting point for further work in the integration of hardware-software co-design with time critical systems. Not only this, it opens questions on each sub-area involved during its development. Although some of these questions were already presented in previous chapters, this section shows a summary of some possible directions for future work.

7.3.1 Resource Utilisation and FPGAs

Currently, each task that is moved to hardware has a dedicated processor to execute on. The incorporation of clique partitioning techniques [84] or other clustering techniques (e.g. [78]) would allow for re-utilisation of hardware processors and improvement of the pre-partitioning phase. These techniques would be used to group related tasks so that parts of the design could be used by tasks allocated to the same processor.

Another research direction based on the same idea is the use of dynamic re-configurable Field Programmable Gate Arrays (FPGAs). Different tasks could use time multiplexing to execute in the same FPGA. After a task execution, re-configuration would make the device suitable to execute another different task. The time separation necessary to allow re-configuration can be guaranteed by incorporating "dummy" segments to the design. For example, if two temporal instances A and B of different tasks are to be executed consecutively in a FPGA, a dummy segment D representing the required re-configuration time could be added between A and B together with the relations $A \mapsto D$ and $D \mapsto B$. The scheduler would then be used to verify if such a configuration is feasible. Here again, clustering techniques could decrease reconfiguration time and device size, by promoting the grouping of tasks which present similar designs.

Still in the same line of thought, the approach used here for system partitioning only considers moving tasks to hardware, since this is the objective in hardware-software partitioning. However, techniques used in software allocation over distributed systems could be useful at this phase, mainly when prototyping is the goal.

7.3.2 Stochastic Analysis

Only hard-real-time systems were considered. As such, soft deadline tasks are not properly considered. Stochastic analysis or queue theory can be used in order to provide limits on the minimum necessary processor time that need to be made available to those tasks (examples can be found in [72]). For this purpose, analysis based on stochastic Petri nets could be useful, since it uses a similar framework as the one presented here.

Probabilistic approaches have been used in conjunction with Petri nets for some time [90], and due to the similar framework should not present great difficulty for incorporation in the environment shown here.

7.3.3 Framework Expansion

The system output is currently presented as separate task sets for hardware and software and it is user's responsibility to translate those specifications to formats suitable to be used by compilers and hardware synthesis tools. It is also user's responsibility to provide accurate estimations to be used as worst-case execution times for both hardware and software.

The utilisation of automatic synthesis techniques, such as [96] would be of great value to tackle both problems mentioned above. Also, the use of formal parallel languages for action and predicate textual specification on transitions would allow for automatic translations between net and textual specifications whenever necessary. It is intended to experiment changing the language used for representing transition functionality to a formal one, possibly a subset of Occam 2, in order to perform automatic refinement.

List partitioning provides a very simple and sometimes ineffective way of choosing tasks to move to hardware, since it does not consider all possibilities and can easily become stranded in local minima. Other techniques can be employed over the candidates chosen by the methods developed here, such as simulated annealing [42] and multi-stage clustering [8].

At the same time, only time-related parameters were considered in this dissertation. Other parameters targeted to deal with area, power, etc., should be incorporated to the cost function.

Experience with methodology usage has shown that another sort of abstraction, even if only visual (such as in Section 3.3.5, and [62, 65], would be desired to encapsulate related tasks and process nets.

The current environment expansion towards the creation of a customisable framework where different tools can be incorporated in order to deal with different aspects of embedded systems design is being considered. As mentioned above, improvements and additions in phases such specification, analysis, prototyping, and synthesis, are part of the future work plans. Furthermore, research on the use of fixed-priority dynamic scheduling together with pre-runtime scheduling is also under consideration.