Scheduling and Optimisation of Heterogeneous Time / Event-Triggered Distributed Embedded Systems

Traian Pop

ISBN 91-7373-676-7 ISSN 0280-7971
PRINTED IN LINKÖPING, SWEDEN
BY LINKÖPING UNIVERSITY

To my parents

Abstract

Day by day, we are witnessing a considerable increase in number and range of applications which entail the use of embedded computer systems. This increase is closely followed by the growth in complexity of applications controlled by embedded systems, often involving strict timing requirements, like in the case of safety-critical applications. Efficient design of such complex systems requires powerful and accurate tools that support the designer from the early phases of the design process.

This thesis focuses on the study of real-time distributed embedded systems and, in particular, we concentrate on a certain aspect of their real-time behavior and implementation: the time-triggered (TT) and event-triggered (ET) nature of the applications and of the communication protocols. Over the years, TT and ET systems have been usually considered independently, assuming that an application was entirely ET or TT. However, nowadays, the growing complexity of current applications has generated the need for intermixing TT and ET functionality. Such a development has led us to the identification of several interesting problems that are approached in this thesis. First, we focus on the elaboration of a holistic schedulability analysis for heterogeneous TT/ET task sets which interact according to a communication protocol based on both static and dynamic messages. Second, we use the holistic schedulability analysis in order to guide decisions during the design process. We propose a design optimisation heuristic that *partitions* the task-set and the messages into the TT and ET domains, *maps* and *schedules* the partitioned functionality, and optimises the *communication protocol parameter*s. Experiments have been carried out in order to measure the efficiency of the proposed techniques.

Acknowledgements

THIS THESIS would have not been possible without the generous support and patient guidance of my supervisors: Petru, whose energy and dedication always amazed me, and Zebo, the right person always at the right time and at the right place.

The working environment here at IDA is probably one of the best I'll ever see in my life. To every staff member who contributed in one way or another to the smooth publication process of this thesis, a sincere thank you. My ESLAB colleagues deserve a special mention, for being some of my closest friends during the last years.

Last, but not least, I would like to thank Ruxandra and to the members of my family, especially to my parents and to my sister, who have always been there for me. Thank you.

Traian Pop
Linköping, Spring 2003

Contents

Chapter 1
Introduction

THIS THESIS DEALS with specific issues related to the system-level design of distributed embedded systems implemented with mixed, event-triggered (ET) and time-triggered (TT) task sets which communicate over bus protocols consisting of both static (ST) and dynamic (DYN) phases. We have focused on the scheduling of heterogeneous TT/ET systems and we have studied the factors which influence the efficiency of the scheduling process. We have also identified several optimisation problems specific for this type of heterogeneous systems, and we have approached these problems in the context of design optimisation heuristics.

This chapter starts by presenting the framework of our thesis, namely the area of distributed embedded real-time systems. We make a short introduction to event-triggered and time-triggered execution of tasks, as well as a brief description of static and dynamic transmission of messages. We introduce both homogeneous and heterogeneous TT/ET distributed embedded systems and we focus on the later ones, as they constitute the motivation behind this work.

Analysis and design of distributed embedded systems has been and will be a prolific area of research, considerably boosted

by the variety of communication protocols which are involved. This thesis is not the first and definitely not the last contribution in this area. In Section 1.3, the reader is acquainted with other work related to the one presented in our thesis, while in Section 1.4 we outline our contributions to the field of analysis and design of embedded real-time systems.

Finally, Section 1.5 is a feedforward to the following chapters.

1.1 Design Flow of Distributed Embedded Systems

Today, *embedded systems* find their place in more and more applications around us, starting with consumer electronics and appliances and ending with safety critical systems in applications such as aerospace/avionics, railway, automotive industry, medical equipment, etc. Quite often, such systems are also *real-time systems*, as they are constrained to perform certain tasks in a limited amount of time; failure to comply with the timing requirements leads to consequences whose gravity can vary from almost imperceptible loss of quality in an MPEG decoder, up to catastrophic events, like fatal car crashes when braking and air-bag systems fail to react in time. Depending on the nature of the timing constraints real-time systems can be classified into *soft* real-time systems, in which deadlines can be occasionally missed without the system reaching an intolerable state, and *hard* real-time systems, in which missing a deadline is intolerable because of its possible consequences [Kop97]. This thesis focuses on hard real-time systems.

Designing a hard real-time embedded system requires procedures for guaranteeing that all deadlines will be met. If such guarantees cannot be provided, then the system is considered *unschedulable* and most likely, its implementation will not meet the requirements in terms of timeliness.

The continuous increase in range and number of applications

entailing the use of embedded systems [Tur99] is closely followed by an increase in complexity of the applications themselves. Complex environments need more and more complex control embedded systems. The growing complexity of real-time embedded systems is also considerably increased by their heterogeneous nature, which goes along several dimensions like:

- applications can be data or control intensive;
- the system functionality implies both hard and soft timing requirements;
- the controlled environment can generate discrete or continuous stimuli;
- components inside an embedded computer system can interact among themselves using different synchronisation mechanisms;
- hardware implementations are based on heterogeneous architectures in which one can find application-specific instruction processors (ASIPs), digital signal processors (DSPs), general purpose processors, protocol processors, application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), etc., all organised in various topologies and interconnected by diverse shared buses, point-to-point links or networks;
- the system includes both analog and digital components.

In this thesis, we have studied another dimension of heterogeneity, resulted from the two different approaches to the design of real-time embedded systems:

- the *time-triggered* approach, in which the processing and communication activities are initiated at predetermined points in time;
- the *event-triggered* approach, in which activities happen when a significant change of state in the system occurs.

As we will see in Chapter 2, the systems which we consider support both time-triggered and event-triggered processing and communication activities.

In Figure 1.1 we present a system-level design flow (adapted from [Ele02]) that starts from a high-level system specification, which may be expressed in several languages, including natural language. The system specification is later refined into an abstract formal model (which can be captured in one or several modelling languages). Starting from the system model, the methodology follows a design exploration stage in which various system architectures are selected, different ways to map the functionality on the available resources are evaluated, and several alternatives for scheduling and synthesis of the communication parameters are examined, so that in the end, the resulted model of the system will meet the requirements imposed for the current design.

In Figure 1.1 we marked with dark rectangles the phases in the design process which are covered in this thesis. First, we developed a method for scheduling and schedulability analysis of the activities in a heterogeneous TT/ET embedded system. This analysis method is then used for guiding the design process, and in particular we concentrated on the problems of *mapping* of functionality, *communication synthesis* and the specific aspect of *partitioning* the functionality into TT and ET activities.

1.2 Heterogeneous ET/TT Systems

In this thesis, we consider heterogeneous embedded systems in the sense that they consist of both time-triggered (TT) and event-triggered (ET) activities. In this section, we present the characteristics of such activities, the typical mechanisms used for implementation and the advantages and disadvantages inherent to each approach.

1.2.1 EVENT/TIME-TRIGGERED TASK EXECUTION

We start by describing first the execution mechanism of tasks in

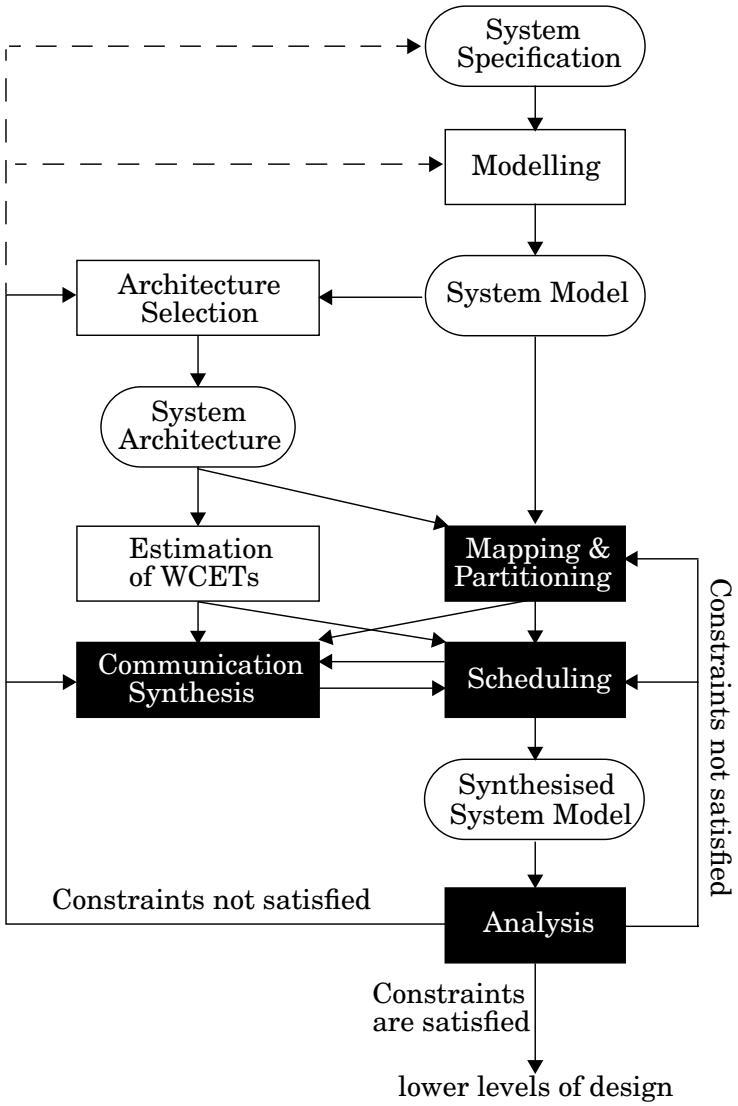


Figure 1.1: System Level Design Flow

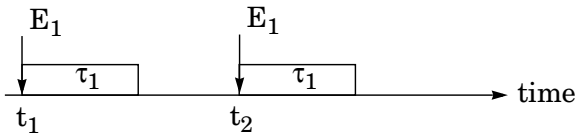


Figure 1.2: ET Task Execution

an ET and then in a TT system. In this thesis we consider that the functionality of the system is decomposed into a set of interacting tasks (Section 2.4). A *task* is defined as "a computation that is executed by the CPU in a sequential fashion" [But97].

1.2.1.1 EVENT-TRIGGERED TASKS

In the event-triggered approach, the execution of a task is initiated by the occurrence of a certain event which is related to a change in the system state. For example, in Figure 1.2, task τ_1 is initiated by event E_1 which appears at times t_1 and t_2. If the resources needed by task τ_1 are available at moment t_1 (for example, the CPU is idle), then task τ_1 starts its execution. The mechanism behaves similarly at moment t_2.

Usually, the system functionality is composed of several tasks and their execution might lead to resource conflicts, like in the case when two tasks are simultaneously ready for execution and only one of them can make use of the processing capabilities of the system. Typically, such conflicts are solved by assigning priorities to tasks and executing the task with the highest priority. We present below one of the simplest and most common approaches, the fixed priority approach, in which the priorities are statically assigned offline to tasks and do not change at run time.

In order to implement a fixed priority policy for task execution, a real-time kernel has a main component called *scheduler* which has two main responsibilities:

- to maintain/update the prioritised queue of ready tasks;
- to select from the queue and execute the ready task with the highest priority.

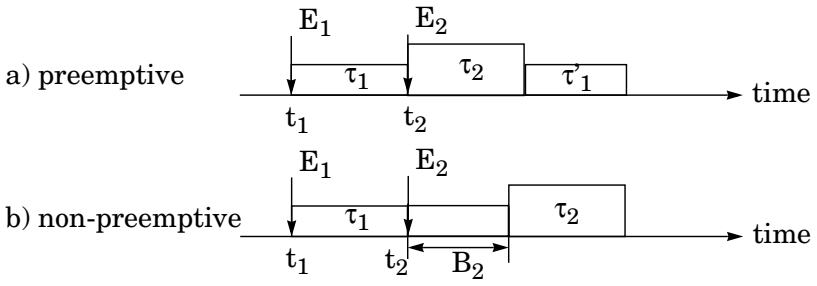


Figure 1.3: Concurrent ET Execution of Tasks

The timeline in Figure 1.3 presents how two conflicting ET tasks are executed by such a real-time kernel. In the first case, the kernel implements a preemptive policy for task execution. When task τ_2 is initiated by the occurrence of event E_2, task τ_1 will be interrupted because it has a lower priority than the priority of task τ_2. Task τ_1 is placed in the ready queue and it will resume its execution only after task τ_2 finishes. In the second case, the execution is non-preemptive and task τ_2 has to wait until task τ_1 finishes execution. In this case, even if task τ_2 has a higher priority than task τ_1, it will be blocked for an amount of time B_2 and it will have to stay in the ready queue until a subsequent activation of the scheduler will find the processor available.

The advantages of the event-triggered approach are its flexibility and an efficient usage of the available resources. However, taking into consideration the overheads related to task switching, scheduler activation, etc. considerably increases the difficulty of the schedulability analysis for such types of systems.

1.2.1.2 TIME-TRIGGERED TASKS

In a time-triggered system, the execution of tasks is initiated at pre-determined moments in time. The main component of the real-time kernel is the time interrupt routine and the main control signal is the clock of the system. The information needed for task execution is stored in a data structure called *schedule table*,

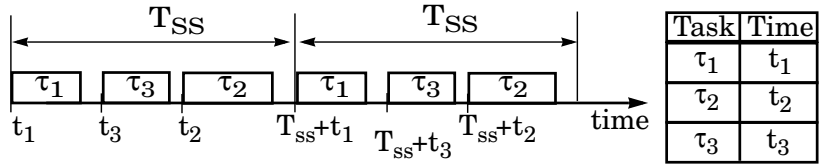


Figure 1.4: Time Triggered Execution of Tasks

where each task has a pre-assigned start time. The schedule table is obtained through a static scheduling algorithm, which is executed off-line and which eliminates the possible conflicts between tasks by imposing appropriate start times.

For example, in Figure 1.4, we consider three periodic tasks, each task being executed with period *T*. The schedule table on the right side of the figure shows that the executions of the three tasks τ_1, τ_2 and τ_3 are started at moments t_1, t_2 and t_3. Each start time in the table is computed offline in such a way that the execution of a task is finished before the next start time stored in the schedule table. After a certain time T_{SS}, called the period of the static cyclic schedule, the kernel performs again the same sequence of decisions.

The period T_{SS} is computed as the least common multiple of the periods of the individual tasks in the system. The case presented above is a very particular one, as all three tasks have the same period *T,* which gives a perfectly harmonised system, and therefore $T_{SS} = T$. However, one may notice that the size of the schedule table increases substantially if the task periods are not harmonised.

Also, a time-triggered system based on a static schedule table has a low flexibility and is usually inappropriate for dynamic environments for which it provides an inefficient processor utilisation.

However, the time-triggered approach has several important advantages. Being highly predictable, deterministic, easy to be validated and verified, it is particularly suitable for safety-criti-

cal applications [Kop97].

1.2.2 EVENT/TIME-TRIGGERED TRAFFIC

The previous section presented activation mechanisms of tasks in a real-time system. We continue with a similar discussion but in the context of communication activities in architectures based on broadcast buses.

There are two main features characteristic to broadcast buses:

- all nodes connected to the communication channel (the bus) receive the same messages; and
- only one node can send messages at a time on the bus. This feature enforces the usage of a bus arbitration method.

Below we present some of the main bus access strategies in use today:

a. Collision Sense Multiple Access (CSMA), in which nodes are enabled to identify if there is any activity on the bus and, if there is none, then the nodes may send. It may be the case that more than one node identifies no activity and sends messages on the bus at the same time, leading to the apparition of collisions between messages. For this reason, CSMA is usually combined with another strategy for dealing with collisions, such as:
   - collision detection (CSMA/CD), where each sending node withdraws when collisions are sensed on the bus and tries to send again later, after a random time (for example, the Ethernet protocol [IEEE98]).
   - collision avoidance (CSMA/CA), where messages have unique priorities that are used for non-destructive bitwise arbitration. A node sending a message with a certain priority stops the transmission when it detects there is another message with a higher priority being sent on the bus (for example, the CAN protocol [Bos91]).

b. Token passing, in which a node is allowed to send messages on the bus only if it is in the possession of a message contain-

ing a piece of information called token (for example, the token bus [IEEE83]).

c. **Mini-slotting**, in which each node has a uniquely associated wait time relative to the start of the bus cycle. If the waiting time has passed and no activity is sensed on the bus, then the node can send messages on the bus (for example, ARINC 629 [ARI629], Byteflight [Ber03], and FlexRay [Fle03]).

d. **Time Division Multiple Access (TDMA)**, in which each node has a pre-assigned time slot for transmitting messages. The access scheme is cyclic and allows each node to send messages periodically without any interference from other nodes (for example TTP/C [TTP01C], FlexRay [Fle03]).

e. **Central Master**, in which a node can send messages on the bus only at the request of another node, which is the central master and which is the only one that can initiate the communication protocol (for example, LIN [LIN00], TTP/A [TTP01A]).

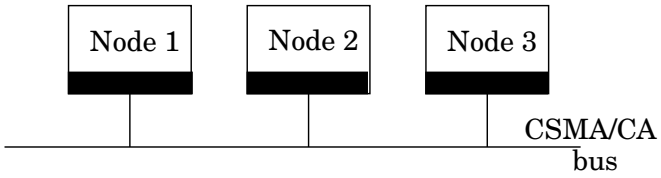
In the following two sub-sections, we discuss two approaches to communication in distributed real-time systems:

1. Dynamic communication (DYN), in which the communication activities are triggered dynamically, in response to an event.
2. Static communication (ST), in which the communication activities are triggered at predetermined moments in time. For such a case, each node in the system knows (from design time) exactly when and which messages are sent on the bus, as well as how long their transmission takes.

1.2.2.1 DYNAMIC COMMUNICATION

In the case of DYN communication, the trigger which initiates the process of sending a message is the generation of the message itself (by the sending task).

We will give an example of how messages are sent over the



Identifiers	Status of the Bus (as seen from each node)
m_1: 11010010	1 1 1 <- $Node_1$ stops transmitting
m_2: 11100110	1 1 1 0 1 <- $Node_2$ stops transmitting
m_3: 11101000	1 1 1 0 1 0 0 0 <- $Node_3$ starts transmitting m_3
	→ time

Figure 1.5: CSMA/CA Bus - Bitwise Arbitration

CAN bus, which is one of the most used event-triggered communication approaches ([Bos91]). The CAN protocol is based on a CSMA/CA arbitration policy, and for this purpose each message in the system has a unique identifier associated to it. Whenever the communication controller in a node receives a message to be sent on the bus, it will have first to wait until the bus is available. When no activity is identified on the bus anymore, the message will be sent, preceded by its unique identifier. The identifier of a message acts like a priority, in the sense that if there are several nodes which transmit at the same time on the bus, only the message with the highest priority will go through and the other ones will have to wait the next moment when the bus becomes available. The collisions between messages whose transmission start at the same time are avoided by a non-destructive bitwise arbitration based on the message identifier.

The collision avoidance mechanism is illustrated in Figure 1.5, where three messages m_1, m_2 and m_3 are simultaneously generated on three different nodes. All three messages start being transmitted at the same time. Each message is preceded on the bus by the sequence of several bits representing its priority. The bus is usually hardwired in such a way that it will

always have the same value in the case a collision appears. This means that if two nodes transmit two different bits simultaneously, then only the *dominant* bit will be sensed on the bus. The example in Figure 1.5 considers the case where the dominant bit is 1, and as a result, after 3 bits have been sent on the bus, the first node gives up the transmission, as it sensed a higher priority on the bus than the one sent by itself. The second node gives up after transmitting 5 bits. Having the highest value for the identifier, the message transmitted by the third node will go undeterred on the bus, while messages m_1 and m_2 will be resent only after transmission of m_3 will finish (of course, the bus access mechanism will decide again which of the remaining messages goes first).

1.2.2.2 STATIC COMMUNICATION

In Section 1.2.1.2 we presented the time-triggered execution of tasks. Similarly, static (ST) communication activities are initiated at predetermined moments of time. A consistent behavior of such a distributed multiprocessor time-triggered system requires that the clocks in all the nodes in the system are synchronised to provide a global notion of time [Kop97]. Such a synchronisation can be efficiently achieved through the communication protocol.

In this section, we detail the time-triggered communication mechanism as it appears in the case of a TDMA bus. As we already mentioned, in the case of a TDMA bus the bandwidth is divided into timeslots and each such slot is assigned offline to a node in the system. During its timeslot, a node has the exclusive right to send messages on the bus. At run-time, if a node has a message to send, it will have to wait until the system time has advanced to the start of its pre-assigned slot. The periodic sequence in which the timeslots are ordered represents a TDMA round.

For example, in Figure 1.6, one can see a distributed system

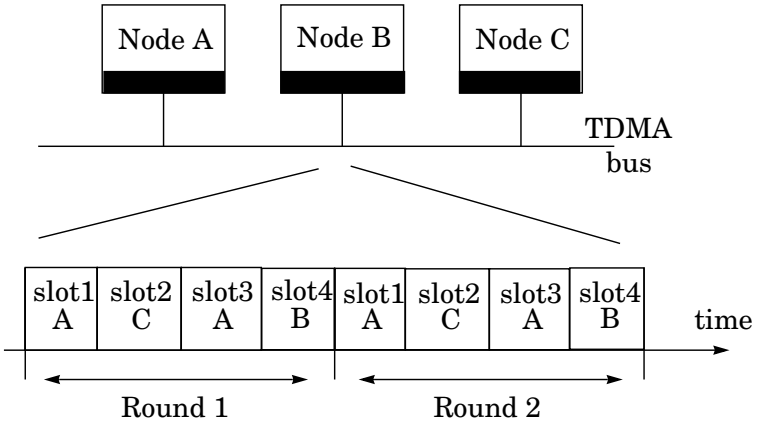


Figure 1.6: TDMA Bus

with three nodes connected to a TDMA bus. The bus cycle is composed of four slots, each slot associated to a node. $Node_A$, for example, can send messages only during $slot_1$ and $slot_3$ of each TDMA round, $Node_B$ can send only during $slot_4$, while $Node_C$ can send only during the second slot of each round. In this way it is guaranteed that only one node transmits on the bus at a time. The TDMA round in the example consists of the sequence of slots 1, 2, 3 and 4.

A typical TDMA based communication protocol is the Time-Triggered Protocol (TTP) [TTP01C]. In the case of TTP, every node stores locally the information related to each of the messages in the system: sender/receiver, starting time of transmission, message length, etc. A node will send a message on the bus whenever the global current time reaches one of the start time values which are stored locally. For example, in Figure 1.7, $Node_A$ starts sending a message m_{AB} at time t_1 relative to the start of each bus round, during its pre-assigned slot in the second round, according to the information stored locally. At the same time, the communication controller in $Node_B$ will know from its own local table that at time t_1 it will have to start read-

Message ID	Start Time	Length	Sender	Receiver
m_{AB}	t_1	C_1	Node A	Node B
m_{BA}	t_2	C_2	Node B	Node A

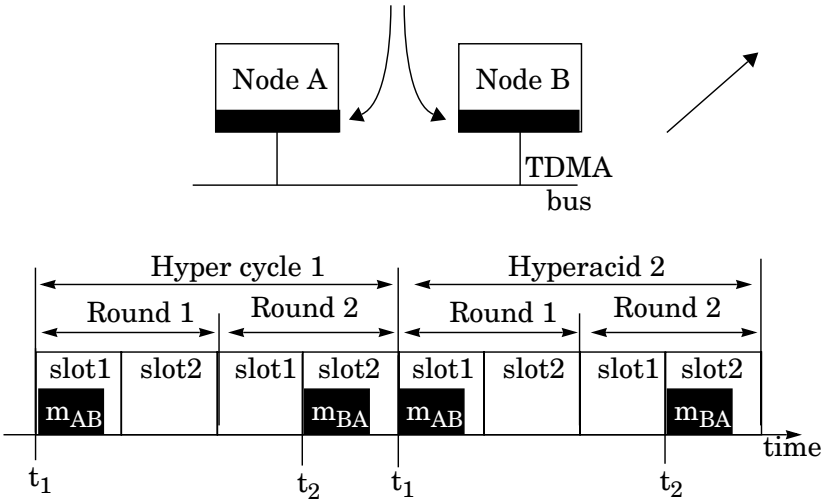


Figure 1.7: Statically Scheduled TT Communication

ing message m_{AB}. At time t_2, another message is scheduled to be transmitted on the bus from $Node_B$ towards $Node_A$. The static schedule illustrated in Figure 1.7 expands along two bus cycles, called *rounds*, and the sequence of such two consecutive rounds forms a *hyper cycle*. The static schedule stored locally in each node is repeated periodically with a period equal to the length of such a hyper cycle.

It is largely accepted that the static properties inherent to the TDMA communication considerably diminish the flexibility of the system. Unless bandwidth is reserved from the design time, adding another sending node in the system requires a reconfiguration of the bus round, which usually triggers many other updates and validations of the system design.

However, the determinism associated with the TDMA commu-

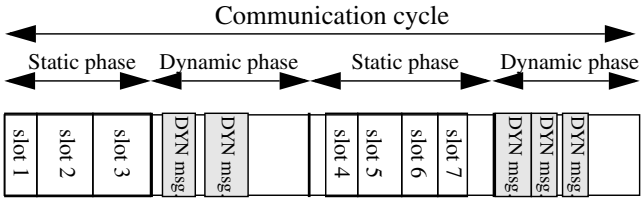


Figure 1.8: Mixed Communication Cycle

nication has several major advantages: timing properties of the system are easily guaranteed, system composability is straightforward when extensions are planned, etc.[Kop97].

1.2.2.3 MIXED PROTOCOLS

Nowadays, protocols which support both time-triggered and event-triggered communication are being developed and placed on the market. Examples in this sense are Flexray [Fle03], WorldFIP [Wor03] and FTT-CAN [Ple92]. The main motivation behind their appearance was to provide a bus support which combines the advantages of both ET and TT approaches into powerful and versatile protocols. One of the main advantages of such protocols is represented by the combination of flexibility and determinism, making them appropriate for implementation of flexible real-time systems which have dynamic requirements as well as timing constraints.

In order to avoid the interferences between ET and TT communication, interference which may have a negative impact on the properties of the TT messages, such a mixed protocol has to enforce a temporal isolation between the two types of traffic. The most common solution is based on the so called *communication cycle* which is split into TT and ET phases that repeat periodically: TT messages are sent during a TT phase, while ET messages are sent during a ET phase ([Raj93], [Ple92]).

In Figure 1.8, we present a generalised model of such a proto-

col, called Universal Communication Model (UCM [Dem01]), in which the communication cycle contains several *static* (ST) and *dynamic* (DYN) phases. A system based on such a protocol will send the TT messages during ST slots according to a pre-defined TDMA scheme and to an associated static schedule, while the ET messages are packed online into frames and sent during the DYN phases according to an arbitration mechanism (like, for example, CSMA/CA or mini-slotting).

The Universal Communication Model allows for the modeling and exploration of a large range of mixed ST/DYN communication protocols for bus based systems. This is why in this thesis, we model the communication on the bus using UCM (Section 2.2).

1.2.3 HETEROGENEOUS SYSTEMS

There has been a lot of debate in the literature on the suitability of the event-triggered paradigm as opposed to the time-triggered one, for implementation of real-time systems [Aud93], [Kop97], [Xu93]. Several arguments have been brought concerning composability, flexibility, fault tolerance, jitter control or efficiency in processor utilisation. The same discussion has also been extended to the communication infrastructure which can also be implemented according to the time-triggered or event-triggered paradigm.

An interesting comparison of the TT and ET approaches, from a more industrial, in particular automotive, perspective, can be found in [Lön99]. Their conclusion is that one has to choose the right approach depending on the particularities of the scheduled tasks. This means not only that there is no single "best" approach to be used, but also that, inside a certain application the two approaches can be used together, some tasks being time-triggered and others event-triggered.

The growing amount and diversity of functions to be implemented by the current and future embedded applications (like

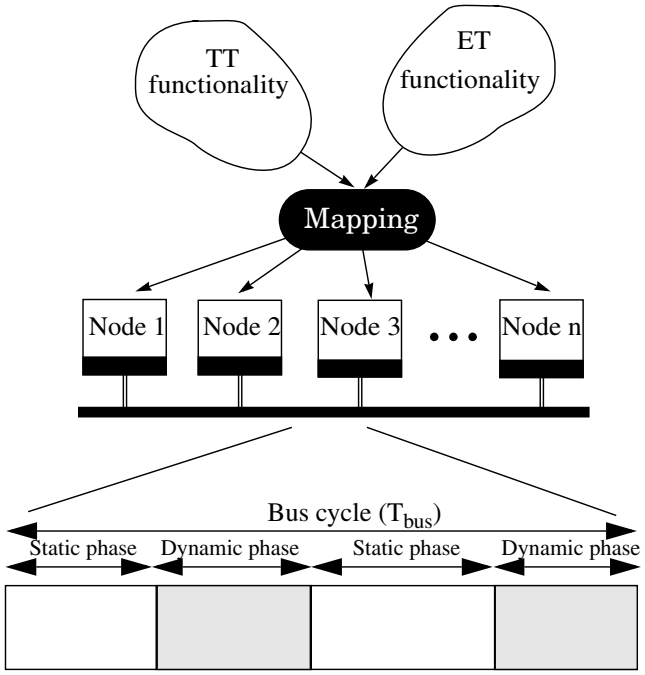


Figure 1.9: Heterogeneous TT/ET Distributed System

for example, in automotive electronics [Koo02]) has shown that, in many cases, time-triggered and event-triggered functions have to coexist on the computing nodes and to interact over the communication infrastructure (see Figure 1.9).

In order to cope with the complexity of designing such heterogeneous embedded systems, only an adequate design environment can effectively support decisions leading in an acceptable time to cost-efficient, reliable and high performance solutions. Developing flexible and powerful tools for the design and analysis of such kind of heterogeneous systems represents the motivation behind the work presented in this thesis.

1.3 Related Work

This section presents an overview of the previous research in the area of analysis and system level design for distributed embedded systems. We concentrate in particular on scheduling and communication synthesis, with focus on the time-triggered and event-triggered aspects.

1.3.1 SYSTEM LEVEL DESIGN

System level design methodology is continuously evolving [Mar00], from ad-hoc approaches based on human designer's experience, to hardware/software codesign, and currently to platform-based design [Keu00] and function-architecture codesign [Bal97], [Lav99], [Tab00].

The design flow presented in Figure 1.1 illustrates only some of the main problems which appear during the system level phases of design. For a deeper insight into system level design aspects with focus on hardware/software trade-offs, the reader is referred to the surveys in [Wol94], [Mic97], [Ern98] and [Wol03].

System modelling has received a lot of attention, as powerful computational models and expressive specification languages are needed in order to capture heterogeneous system requirements and properties at different levels of abstraction [Edw97], [Edw00], [Lav99]. Evaluation of system performance with regard to timing requirements usually starts with static analysis or other means for performance estimation of the functionality [Ern97]. Typical hardware architectures for embedded systems have evolved from simple ones (involving only one processor and one ASIC), to distributed and heterogeneous ones, as described in Section 1.1. Such an evolution has directly increased the complexity of the problems related to architecture selection, mapping, partitioning and scheduling of functionality and has led to the apparition of new approaches like those proposed in [Bec98], [Bli98], [Dav99], [Lee99], [Wol97], [Yen97].

1.3.2 SCHEDULING AND SCHEDULABILITY ANALYSIS OF REAL-TIME SYSTEMS

Task scheduling and schedulability analysis have been intensively studied for the past decades, one of the reasons being the high complexity of the targeted problems [Ull75], [Sta94]. The reader is referred to [Aud95] and [Bal98] for surveys on this topic.

A comparison of the two main approaches for scheduling hard real-time systems (i.e., *static cyclic scheduling* and *fixed priority scheduling*) can be found in [Loc92].

The *static cyclic (non-preemptive) scheduling* approach has been long considered as the only way to solve a certain class of problems [Xu93]. This was one of the main reasons why it received considerable attention. Solutions for generating static schedules are often based on *list scheduling* in which the order of selection for tasks plays the most important role [Coff72], [Jor97] (see also Section 3.5). However, list scheduling is not the only alternative, and branch-and-bound algorithms [Jon97], [Abd99], mixed integer linear programming [Pra92], constraint logic programming [Kuc97], [Eke00], or evolutionary [Sch94] approaches have also been proposed.

For event-triggered tasks, in this thesis we are interested only in static priority based scheduling policies. In the case of *fixed priority (preemptive) scheduling*, determining whether a set of tasks is schedulable involves two aspects:

1. *The assignment of priorities* to system activities, i.e. what priority should be associated with each task and message in the system so that the task set is schedulable.
2. *The schedulability test*, which determines whether all activities in the system will meet their deadlines under the current policy.

In order to solve the problem of assigning priorities to system activities so that the system is schedulable, two main policies have been developed; they both work under restricted assump-

tions, i.e. the task set to be scheduled is composed of periodic and independent tasks mapped on a single processor:

a. rate-monotonic (RM) [Liu73] which assigns higher priorities to tasks with shorter periods; it works under the constraint that task deadlines are identical with task periods.

b. deadline-monotonic (DM) [Leu82] which assigns higher priorities to tasks with shorter relative deadlines; this policy assumes that task deadlines are shorter than task periods.

If, for example, tasks are not independent, then the optimality does not hold anymore for RM and DM policies. Therefore, in [Aud93], the authors proposed an optimal[1] solution for priority assignment in the case of tasks with arbitrary release times. Their algorithm is of polynomial complexity in the number of tasks. However, for the case of multiprocessor/distributed hard real-time systems, obtaining an optimal solution for priority assignment is often infeasible, due to complexity reasons. A solution based on simulated annealing has been proposed in [Tin92], where the authors present an algorithm which simultaneously maps the tasks on processors and assigns priorities to system activities so that the resulted system is schedulable. In order to avoid the large amount of computation time required by such a general-purpose approach, an optimised priority assignment heuristic called HOPA has been suggested in [Gut95], where the authors iteratively compute deadlines for individual tasks and messages in the system, while relying on the DM policy to assign priorities to the tasks. Their algorithm has shown a better efficiency than simulated annealing, both in quality and especially in speed, making it appropriate for being used inside a design optimisation loop which requires many iterations. As an example, HOPA has been adapted for the design optimisation of multi-cluster distributed embedded systems [Pop03b].

1. The algorithm is optimal in the sense that it finds a solution whenever one exists.

For the second aspect of fixed priority scheduling, there are two main approaches for performing schedulability tests:

a. *utilisation based tests*, in which the schedulability criterion is represented by inequations involving processor utilisation and utilisation bounds. However, such approaches are valid only under restricted assumptions [Liu73], [Bin01], [Leu82].

b. *response time analysis*, in which determining whether the system is schedulable or not requires first the computation of the worst-case response time of a task or message. The worst case response time of an activity is represented by the longest possible time interval between the instant when that activity is initiated in the system and the moment when the same activity is finished. If the worst case response time resulted for each task/message is lower or equal than the associated deadline for that activity, then the system is schedulable.

Response time analysis is usually more complex but also more powerful than the utilisation based tests. The main reason for this is because response time analysis can take into consideration more factors that influence the timing properties of tasks and messages in a system.

The response time analysis in [Leh89] offers a necessary and sufficient condition for scheduling tasks running on a mono-processor system, under fixed priority scheduling and restricted assumptions (independent periodic tasks with deadlines equal with periods). In order to increase the range of target applications, relaxing/restricting assumptions is necessary. Moreover, considering the effects of more and more factors that influence the timing properties of the tasks decreases the pessimism of the analysis by determining tighter worst case response times and leading to a smaller number of false negatives (which can appear when a system which is practically schedulable cannot be proven so by the analysis). Over the time, extensions have been offered to response time analysis for fixed priority schedul-

ing by taking into account task synchronisation [Sha90], arbitrary deadlines [Leh90], precedence constraints between tasks [Pal99] and tasks with varying execution priorities [Gon91], arbitrary release times [Aud93], [Tin94c], tasks which suspend themselves [Pal98], tasks running on multiprocessor systems [Tin94a], [Pal98], etc. In [Ric02] and [Ric03], the authors model the multiprocessor heterogeneous systems as components that communicate through event streams and propose a technique for integrating different local scheduling policies based on such event-model interfaces.

1.3.3 COMMUNICATION IN REAL-TIME SYSTEMS

The aspects related to communication in real-time systems are receiving a continuously increasing attention in the literature. Building safety critical real-time systems requires consideration for all the factors that influence the timing properties of a system. For the case of distributed systems, in order to guarantee the timing requirements of the activities in the system, one should consider the effects of communication aspects like the communication protocol, bus arbitration, clock synchronisation, packaging of messages, characteristics of the physical layer, etc. Due to the variety of communication protocols, scheduling and schedulability analysis involving particular communication protocols has become a prolific area of research. Following a similar model for determining task response time under rate monotonic analysis, message transmission times have been analysed for protocols like TTP bus [Kop92], Token Ring [Ple92], [Str89], FDDI [Agr94], ATM [Erm97], [Han97] and CAN bus [Tin94b].

Usually, communication protocols allow either static (time-triggered) or dynamic (event-triggered) services, influencing several levels in the design flow and giving more weight in the design output to either flexibility or time-determinism of the system. As a result, a lot of work has been concentrated on coping with the disadvantages of the TT/ET approaches and on try-

ing to combine their advantages. For example, in [Pop01a] and [Pop01b], the authors present a method for dealing with flexibility in TTP based systems by considering consecutive design stages in a so called *incremental design flow.* In order to combine the advantages of rigid off-line static scheduling with flexible online fixed priority scheduling, in [Dob01a] and [Dob01b], fixed priority scheduling is adapted in such a way that it emulates static cyclic schedules which are generated offline.

In the case of bus-based distributed embedded systems, one of the main directions of evolution for communication protocols is towards mixed protocols, which support both ET and TT traffic. The proponents of the Time-Triggered Architecture showed that TTP can be enhanced in order to transmit ET traffic, while still maintaining time composability and determinism of the system, properties which are normally lost in event-triggered systems [Kop92]. A modified version of CAN, called Flexible Time-Triggered CAN [Alm99], [Alm02], is based on communication cycles which are divided into asynchronous and synchronous windows. Several other mixed communication protocols can be found in [Fuh00],[Wor03], [Fle03].

1.4 Contributions

Our approach considers distributed embedded systems implemented with mixed, event-triggered and time-triggered task sets, which communicate over bus protocols consisting of both static and dynamic phases.

We have considered that the time-triggered activities are executed according to a static cyclic schedule, while the event-triggered activities follow a fixed priority policy, which is preemptive for the execution of tasks and non-preemptive for the transmission of messages. We have modelled the heterogeneous communication protocol using UCM.

The main contributions of this thesis are:

- A holistic schedulability analysis for heterogeneous TT/ET task sets which communicate through mixed ST/DYN communication protocols [PopT02], [PopT03a]. Such an analysis presents two aspects:
  a) It computes the response times of the ET activities while considering the influence of a static schedule;
  b) It builds a static cyclic schedule for the TT activities while trying to minimise the response times of the ET activities.
- The identification of several design issues which are specific to heterogeneous TT/ET embedded systems, along with the motivation for considering them during a design optimisation phase.
- A design optimisation heuristic which simultaneously maps, schedules and partitions the system functionality into ET and TT domains, while also optimises the parameters of the ST/DYN communication protocol [PopT03b].

1.5 Thesis Overview

The next chapter presents the system model we used. In Chapter 3, we present our analysis method for deriving response times of tasks and of messages in a heterogeneous TT/ET system. In Chapter 4, we first discuss some optimisation aspects which are particular to the studied systems, and then we define and solve the design optimisation problem. Finally, in Chapter 5 we draw some conclusions and discuss possible research directions for the future.

Chapter 2
System Model

In this chapter we present the system model which we use during scheduling and design optimisation. First, we briefly describe the hardware architecture and the structure of the bus access cycle. Then, we present the minimal requirements regarding the software architecture for a system which is able to run both event-triggered and time-triggered activities. The last section of this chapter presents the abstract representation which we use for modelling the applications that are assumed to implement the functionality of the system.

2.1 Hardware Architecture

We consider architectures consisting of nodes connected by a unique broadcast communication channel. Each node consists of:

- a *communication controller* which controls the transmission and reception of both ST and DYN messages;
- a *CPU* for running the processes mapped on that particular node;

- *local memories* for storing the code of the kernel (ROM), the code of the processes and the local data (RAM); and
- *I/O interfaces* to sensors and actuators.

Such hardware architectures are common in applications such as automotive electronics, robotics, etc. In Figure 2.1, we illustrate a heterogeneous distributed architecture interconnected by a bus based infrastructure.

2.2 Bus Access

We model the bus access scheme using the Universal Communication Model (see Section 1.2.2.3). The bus access is organised as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static and dynamic phases (Figure 2.1). Static phases consist of time slots, and during a slot only one node is allowed to send ST messages; this is the node associated to that particular slot. During a dynamic phase, all nodes are allowed to send DYN messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism based on priorities

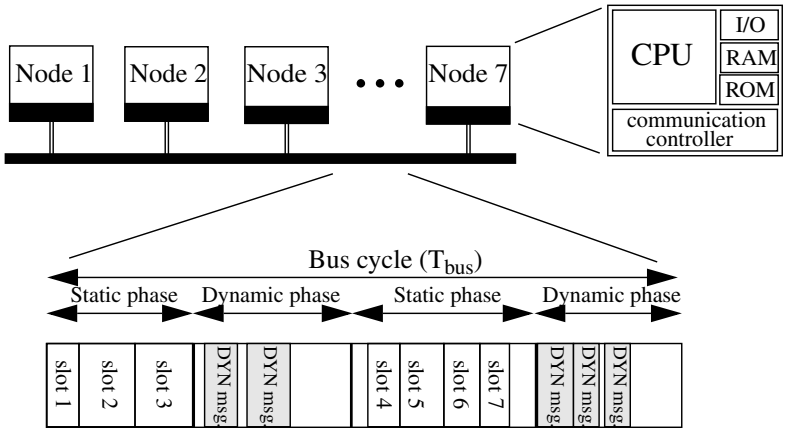


Figure 2.1: System Architecture

assigned to messages. The bus access cycle has the same structure during each period T_{bus}. Every node has a communication controller that implements the static and dynamic protocol services. The controller runs independently of the node's CPU.

2.3 Software Architecture

For the systems we are studying, we have designed a software architecture which runs on the CPU of each node. The main component of the software architecture is a real-time kernel which supports both time-triggered and event-triggered activities. An activity is defined as either the execution of a task or as the transmission of a message on the bus. For the TT activities, the kernel relies on a static schedule table which contains all the information needed to take decisions on activation of TT tasks or transmission of ST messages. For the ET tasks, the kernel maintains a prioritised ready queue in which tasks are placed whenever their triggering event has occurred and they are ready for activation, or when they have been pre-empted.

The real-time kernel will always activate a TT task at the particular time fixed for that task in the schedule table. If at that moment, an ET task is running on that node, that task will be pre-empted and placed into the ready queue according to its priority. If no tasks are active, ET tasks are extracted from the ready queue and are (re)activated. ET tasks can pre-empt each other based on their priority.

The transmission of messages is handled in a similar way: for each node, the sending and receiving times of ST messages are stored in the schedule table; the DYN messages are organised in a prioritised ready queue. ST messages will be placed at predetermined time moments into a bus slot assigned to the sending node. DYN messages can be potentially sent during any dynamic phase. Conflicts due to simultaneous transmission of messages from different nodes are avoided, based on message

priorities, by the communication controllers. We consider that the transmission of messages is non-preemptive, i.e. once the transmission of a DYN message has started, no other message will be sent on the bus until the current transmission finishes. In order to prevent the delay of an ST message by a DYN frame, the DYN messages will be sent only if there is enough time available for that message before the dynamic phase ends.

TT activities are triggered based on a local clock available in each processing node. The synchronisation of local clocks throughout the system is provided by the communication protocol.

2.4 Application Model

We model an application as a set of task graphs. Nodes represent tasks and arcs represent communication (and implicitly dependency) between the connected tasks.

- A task can belong either to the TT or to the ET domain.
- Communication between tasks mapped to different nodes is performed by message passing over the bus. Such a message passing is modelled as a communication task inserted on the arc connecting the sender and the receiver tasks. The communication time between tasks mapped on the same node is considered to be part of the task execution time. Thus, such a communication activity is not modelled explicitly. For the rest of the thesis, when referring to messages we consider only the communication activity over the bus.
- A message can belong either to the static (ST) or to the dynamic (DYN) domain. We consider that static messages are those sent during the ST phases of the bus cycle, while dynamic messages are those transmitted during the DYN

phases.

- All tasks in a certain task graph belong to the same domain, either ET, or TT, which is called the domain of the task graph. The messages belonging to a certain task graph can belong to any domain (ST or DYN). Thus, in the most general case, tasks belonging to a TT graph, for example, can communicate through both ST and DYN messages. In this thesis we restrict our discussion to the situation when TT tasks communicate through ST messages and ET tasks communicate through DYN messages.

- Each task τ_{ij} (belonging to the task graph Γ_i) has a period T_{ij}, and a deadline D_{ij} and, when mapped on node $Proc_k$, it has a worst case execution time $C_{ij}(Proc_k)$. The node on which τ_{ij} is mapped is denoted as $Node_{ij}$. Each ET task also has a given priority $Prio_{ij}$. Individual release times or deadlines of tasks can be modelled by introducing dummy tasks in the task graphs; such dummy tasks have an appropriate execution time and are not mapped on any of the nodes [Ele00a].

- All tasks τ_{ij} belonging to a task graph Γ_i have the same period T_i which is the period of the task graph.

- For each message we know its size (which can be directly converted into communication time on the particular communication bus). The period of a message is identical with that of the sender task. Also, DYN messages have given priorities.

Figure 2.2 shows an application modelled as two task-graphs Γ_1 and Γ_2 mapped on two nodes, $Node_1$ and $Node_2$. Task-graph Γ_1 is time-triggered and task-graph Γ_2 is event-triggered. Data-dependent tasks mapped on different nodes communicate through messages transmitted over the bus, which can be either statically scheduled, like m_1 and m_3, or dynamic, like the messages m_2 and m_4.

In order to keep the separation between the TT and ET

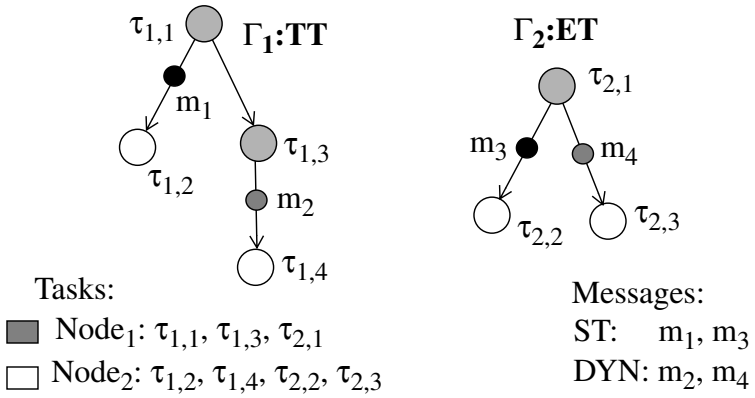


Figure 2.2: Application Model Example

domains, which are based on fundamentally different triggering policies, communication between tasks in the two domains is not included in the model. Technically, such a communication is implemented by the kernel, based on asynchronous non-blocking send and receive primitives (using proxy tasks if the sender and receiver are on different nodes). Such messages are typically non-critical and are not affected by hard real-time constraints.

Chapter 3
Scheduling and Schedulability Analysis of Heterogeneous TT/ET Systems

IN THIS CHAPTER we present an analytic approach for computing task response times and message transmission delays for heterogeneous TT/ET systems.

3.1 Problem Formulation

Given an application and a system architecture as presented in Chapter 2, the following problem has to be solved: construct a correct static cyclic schedule for the TT tasks and ST messages (a schedule which meets all time constraints related to these activities), and conduct a schedulability analysis in order to check that all ET tasks and DYN messages meet their deadlines. Two important aspects should be noticed:

1. When performing the schedulability analysis for the ET

tasks and DYN messages, one has to take into consideration the interference from the statically scheduled TT tasks and ST messages.

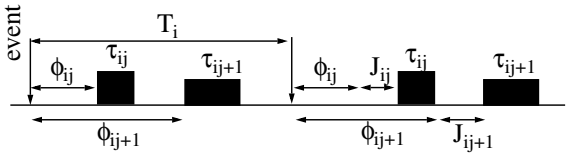
2. Among the possible correct schedules for TT tasks and ST messages, it is important to construct one which favours, as much as possible, the schedulability of ET tasks and DYN messages.

In the next two sections, we will present the schedulability analysis algorithm proposed in [Pal98] for distributed real-time systems and we will show how we extended this analysis in order to consider the interferences induced by an existing static schedule. Section 3.4 presents a general view over our approach for the global scheduling and schedulability analysis of heterogeneous TT/ET distributed embedded systems. In Section 3.5 we present our complete scheduling algorithm, which statically schedules the TT activities while trying to minimise the influence of TT activities onto ET ones. Several alternative implementations of the algorithm have been proposed and compared. Section 3.6 presents the experimental results and evaluations of the proposed heuristics.

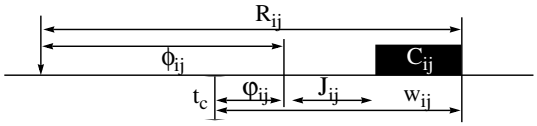
It has to be mentioned that our analysis is restricted, for the moment, to the model in which TT tasks communicate only through ST messages, while communication between ET tasks is performed by DYN messages. This is not an inherent limitation of our approach. For example, schedulability analysis of ET tasks communicating through ST messages has been presented in [Pop00] and [Pop03a].

3.2 Schedulability Analysis of Event-Triggered Task Sets

In this section we briefly describe the schedulability analysis approach presented in [Pal98]. The algorithm is based on computing the worst case response time of ET activities.



a) Tasks with offsets



b) Response time and busy period w for task τ_{ij}

Figure 3.1: Execution Model of the ET Sub-System

An ET task graph Γ_i is activated by an associated event which occurs with a period T_i. Tasks and messages are modelled similarly, by considering the bus as a processing node and accounting for the non-preemptability of the messages during the analysis. Each activity τ_{ij} (task or message) in an ET task graph has an offset ϕ_{ij} which specifies the earliest activation time of τ_{ij} relative to the occurrence of the triggering event. The delay between the earliest possible activation time of τ_{ij} and its actual activation time is modelled as a jitter J_{ij} (Figure 3.1.a). Offsets are the means by which dependencies among tasks are modelled for the schedulability analysis. For example, if in Figure 3.1.a), task τ_{ij+1} is data dependant on task τ_{ij}, then such a relation can be enforced by associating to τ_{ij+1} an offset ϕ_{ij+1} which is equal or greater than the worst case response time R_{ij} of its predecessor, τ_{ij}. In this way, it is guaranteed that task τ_{ij+1} starts only after its predecessor has finished execution.

The response time of an activity τ_{ij} is the time measured from the occurrence of the associated event until the completion of τ_{ij}. Each ET activity τ_{ij} has a best case response time $R_{b,ij}$. The worst case response time R_{ij} of an activity τ_{ij} is determined by

```
01 do
02   Done = true
03   for each transaction Γ_i do
04     for each task τ_ij in Γ_i do
05       for each task τ_ik in Γ_i do
06         if Prio_ik ≥ Prio_ij and Node_ik=Node_ij then
08           for each job p of τ_ij do
09             Consider that τ_ik initiates t_c
10             Compute R_ij^p
11             if (R_ij^p > R_ij^max) then
12               R_ij^max = R_ij^p
13             endif
14           endfor
15         endif
16       endfor
17       if R_ij^max > R_ij then // larger R_ij found
18         R_ij = R_ij^max
19         Done = false
20         for each successor τ_ik of τ_ij do
20           J_ik = R_ij - R_ij^b // update jitters
21         endfor
22       endif
23     endfor
24   endfor
25 while (Done != true)
```

Figure 3.2: Schedulability Analysis Algorithm

creating first a critical instant t_c, which represents the starting point of the worst-case busy window w_{ij}, a time interval which ends when τ_{ij} finishes execution (Figure 3.1.b). During the busy window w_{ij}, $Node_{ij}$ executes only task τ_{ij} or higher priority tasks. The variable φ_{ij} represents the time interval between the critical instant and the earliest time for the first activation of the task after this instant.

Considering a set of data dependent ET tasks, the analysis in [Pal98] computes the worst case response time R_{ij} of a task τ_{ij}, based on the length of its busy period, considering all the critical instants initiated by higher priority activities in Γ_i and by τ_{ij} itself, and all job instances p of τ_{ij} which can appear in the busy window w_{ij}:

$$R_{ij} = max([max(w_{ijk}(p) - \varphi_{ijk} - (p-1)T_i + \phi_{ij})]), \quad \forall k | Prio_{ik} > Prio_{ij}, \forall \text{job p of } \tau_{ij}$$

where $w_{ijk}(p)$ is the length of the worst-case busy window of the p-th job of τ_{ij}, numbered from the critical instant t_c initiated by τ_{ik}; φ_{ijk} is the time interval between the critical instant initiated by τ_{ik}, and the earliest time for the first activation of τ_{ij} after this instant.

The value of $w_{ijk}(p)$ is determined as follows:

$$w_{ijk}(p) = B_{ij} + (p - p_{0,\,ijk} + 1) \cdot C_{ij} + W_{ik}(\tau_{ij}, w_{ijk}(p)) + \sum_{\forall (a \neq i)} W^*_a(\tau_{ij}, w_{ijk}(p)))$$

where B_{ij} represents the maximum interval during which τ_{ij} can be blocked by lower priority activities[2], $W_{ik}(\tau_{ij}, t)$ is the interference from higher priority activities in the same task graph Γ_i at during a time interval of length t relative to t_c, and $W^*_a(\tau_{ij}, t)$ is the maximum interference of activities from other task graphs Γ_a on τ_{ij} during the same interval.

Figure 3.2 represents the pseudocode for the schedulability analysis proposed in [Pal98]. According to this algorithm, the worst case response time R_{ij} of each task τ_{ij} is computed by considering all critical instants initiated by each task τ_{ik} mapped on the same $Node_{ij}$ and with a higher priority than $Prio_{ij}$. According to the same schedulability analysis, jitters are taken into consideration when the algorithm computes the length of the

2. Such blocking can occur at access to a shared critical resource.

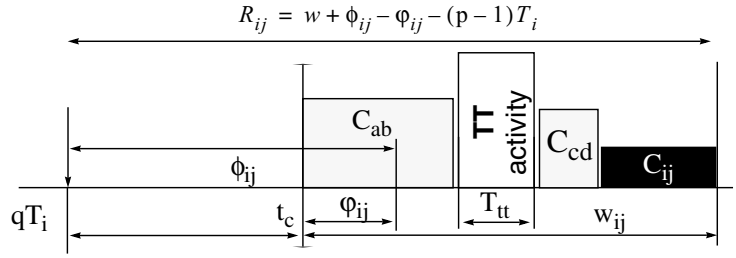
busy windows and, implicitly, the response times of the tasks [Pal98]. This means that the length of the busy window depends on the values of task jitters, which, in turn, are computed as the difference between the response times of two successive tasks (for example, if τ_{ij} precedes τ_{ik} in Γ_i, then $J_{ik} = R_{ij} - R_{b,ij}$, like in lines 20-21 in Figure 3.2). Because of this cyclic dependency (response times depend on jitters and jitters depend on response times), the process of computing R_{ij} is an iterative one: it starts by assigning $R_{b,ij}$ to R_{ij} and then computes the values for jitters J_{ij}, busy windows $w_{ijk}(p)$ and then again the worst-case response times R_{ij}, until the response times converge to their final value.

3.3 Schedulability Analysis of Event-Triggered Activities under the Influence of a Static Cyclic Schedule

Considering the algorithm presented in the previous section as a starting point, we have to solve the following problem: compute the worst case response time of a set of ET tasks and DYN messages by taking into consideration:

- The interference from the set of statically scheduled tasks.
- The characteristics of the communication protocol, which influence the worst case delays induced by the messages communicated on the bus.

As a first step towards the solution of the problem, we introduce the notion of *ET demand* associated with an ET activity τ_{ij} as the amount of CPU time or bus time which is *demanded* only by higher priority ET activities and by τ_{ij} itself during the busy window w_{ij}. In Figure 3.3, the ET demand of the task τ_{ij} during the busy window w_{ij} is represented with $H_{ij}(w_{ij})$, and it is the sum of worst case execution times for task τ_{ij} and two other higher priority tasks τ_{ab} and τ_{cd}. During the same busy period w_{ij}, we define the *availability* as the processing time which is not



For interval $[t_c, t_c+w_{ij}]$ { ET availability: $A^q_{ij}(w_{ij}) = w_{ij} - T_{tt}$; ET demand: $H_{ij}(w_{ij}) = C_{ij} + C_{ab} + C_{cd}$

Figure 3.3: Availability and Demand for a Given Time Interval

used by statically scheduled activities. In Figure 3.3, the CPU availability for the interval of length w_{ij} is obtained by substracting from w_{ij} the amount of processing time needed for the TT activities.

During a busy window w_{ij}, the *ET demand* H_{ij} of a task τ_{ij} is equal with the length of the busy window which would result when considering only ET activity on the system:

$$H_{ij}(w_{ij}) = B_{ij} + (p - p_{0,ijk} + 1) \cdot C_{ij} + W_{ik}(\tau_{ij}, w_{ij}) + \sum_{\forall (a \neq i)} W^*_a(\tau_{ab}, w_{ij})$$

During the same busy window w_{ij}, the availability A_{ij} associated with task τ_{ij} is:

$$A_{ij}(w_{ij}) = min\left[A^q_{ij}(w_{ij})\right], q = 0, \frac{LCM(T_i, T_{SS})}{T_i}$$

where $A^q_{ij}(w)$ is the total available CPU-time on $Node_{ij}$ in the interval $[q\,T_i + \phi_{ij} - \varphi_{ijk}, q\,T_i + \phi_{ij} - \varphi_{ijk} + w_{ij}]$, T_i is the period of Γ_i, and T_{SS} is the period of the static schedule (see Section 3.5). The value of $A^q_{ij}(w)$ is computed using the following equation, which substracts from the given time interval of length w_{ij} those time intervals which are used for execution of TT activities:

```
01 w_ij = p • C_ij + B_ij
02 do
03    Compute demand H_ij(w_ij)
04    Compute availability A_ij(w_ij)
05    if H_ij(w_ij) > A_ij(w_ij) then
06        w_ij = H_ij(w_ij) - A_ij(w_ij)
07    endif
08 while H_ij(w_ij) ≥ A_ij(w_ij)
09 return w_ij
```

Figure 3.4: Determining the Length of the Busy Window

$$A_{ij}^{q}(w_{ij}) = w_{ij} - \sum H_{k}^{TT},$$

where H_k^{TT} represents the duration of the k-th TT activity which takes place inside the interval under analysis.

Figure 3.3 presents how the *availability* $A^q_{ij}(w)$ and the *demand* $H_{ij}(w)$ are computed for a task τ_{ij}: the busy window of τ_{ij} starts at the critical instant $q\ T_i + t_c$ initiated by task τ_{ab} and ends at moment $qT_i + t_c + w_{ij}$, when both higher priority tasks (τ_{ab}, τ_{cd}), all TT tasks scheduled for execution in the analysed interval, and τ_{ij} have finished execution.

The discussion above is, in principle, valid for both ET tasks and DYN messages. However, there exist two important differences. First, messages do not pre-empt each other, therefore, the demand equation is modified so that it will not consider the time needed for the transmission of the message under analysis (once the message has gained the bus it will be sent without any interference [Ple92]). Second, the availability for a message is computed by substracting from w_{ij} the length of the ST slots which appear during the considered interval; moreover, because a DYN message will not be sent unless there is enough time before the current dynamic phase ends, the availability is further decreased with C_A for each dynamic phase in the busy window (where C_A is the transmission time of the longest DYN mes-

sage). This modifications can be noticed in the equations presented below, which are used to determine the values of availability and demand for messages:

$$H_{ij}(w_{ij}) = B_{ij} + (p - p_{0,\,ijk}) \cdot C_{ij} + W_{ik}(\tau_{ij}, w_{ij}) + \sum_{\forall(a \neq i)} W^{*}_{a}(\tau_{ab}, w_{ij})$$

$$A^{q}_{ij}(w_{ij}) = w_{ij} - m \cdot C_{a} - \sum H^{TT}_{k},$$

where m represents the number of DYN phases in the interval under analysis, while H^{TT}_{k} represents the length of the k-th ST phase during the same interval.

Our schedulability analysis algorithm determines the length of a busy window w_{ij} for an ET task or DYN message by identifying the appropriate size of w_{ij} for which the ET demand is satisfied by the availability: $H_{ij}(w_{ij}) \leq A_{ij}(w_{ij})$ (Figure 3.4). This procedure for the calculation of the busy window is included in the iterative process for calculation of response times, presented in the previous subsection (lines 09-10 in Figure 3.2). It is important to notice that this process includes both tasks and messages and, thus, the resulted response times of the ET tasks are computed by taking into consideration the delay induced by the bus communication.

After performing the schedulability analysis, we can check if $R_{ij} \leq D_{ij}$ for all the ET tasks. If this is the case, the set of ET activities is schedulable. In order to drive the global scheduling process, as it will be explained in the next section, it is not sufficient to test if the task set is schedulable or not, but we need a metric that captures the "degree of schedulability" of the task set. For this purpose we use the function *DSch,* similar with the

one described in [Pop00]:

$$\text{DSch} = \begin{cases} f_1 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} max(0, R_{ij} - D_{ij}) \text{ , if } f_1 > 0 \\ f_2 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} (R_{ij} - D_{ij}) \text{ , if } f_1 = 0 \end{cases}$$

where N is the number of ET task graphs and N_i is the number of activities in the ET task graph Γ_i.

If the task set is not schedulable, there exists at least one task for which $R_{ij} > D_{ij}$. In this case, $f_1 > 0$ and the function is a metric of how far we are from achieving schedulability. If the set of ET tasks is schedulable, $f_2 \leq 0$ is used as a metric. A value $f_2 = 0$ means that the task set is "just" schedulable. A smaller value for f_2 means that the ET tasks are schedulable and a certain amount of processing capacity is still available.

Now, that we are able to perform the schedulability analysis for the ET tasks considering the influence from a given static schedule of TT tasks, we can go on to perform the global scheduling and analysis of the whole application.

3.4 Global Scheduling and Schedulability Analysis Strategy

Figure 3.5 illustrates our strategy for scheduling and schedulability analysis of heterogeneous TT/ET distributed embedded systems: the activities to be scheduled are the TT and ET task graphs, consisting of TT tasks/ST messages and ET tasks/DYN messages respectively. The TT activities are statically scheduled and, as an output, a static cyclic schedule will be produced. Similarly, the worst case response times of the ET activities are determined using the schedulability analysis presented in the previous section. As a result, the system is considered schedulable if the static schedule is valid and if the ET activities are

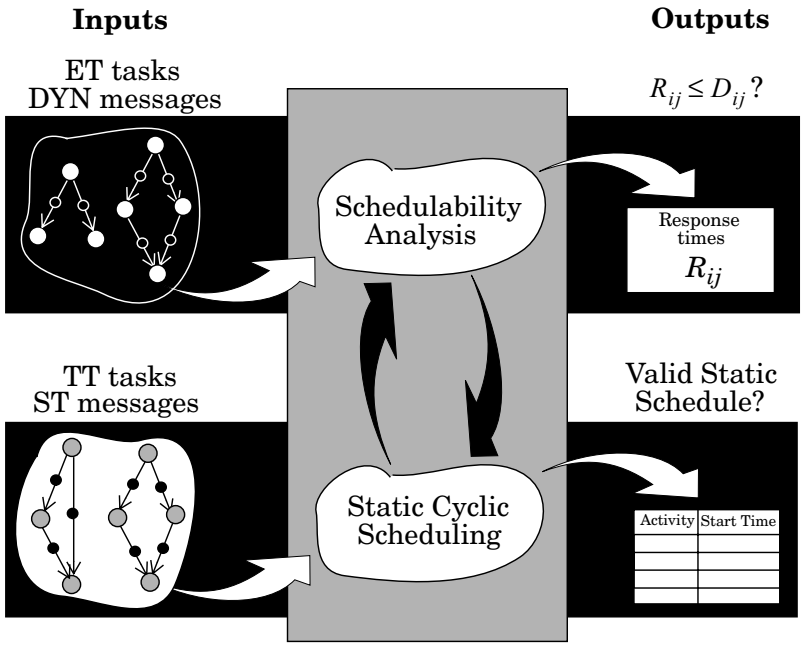


Figure 3.5: Scheduling and Schedulability Analysis for Mixed TT/ET Distributed Embedded Systems

guaranteed to meet their deadlines. For the case of a mixed TT/ET system, building a static cyclic schedule for the TT activities has to take into consideration both the characteristics of the mixed ST/DYN communication protocol and our assumption that execution of TT tasks is non-preemptible, while the execution of an ET task can be interrupted either by a TT task or by another ET task which has a higher priority. This means that the static schedule will have not only to guarantee that TT activities meet their deadlines, but also that the interference introduced from such a schedule will not increase in an unacceptable way the response times of ET activities. In conclusion, an efficient scheduling algorithm requires a close interaction between the static scheduling of TT activities and the schedulability analysis of the ET activities.

```
01 Assign priorities to tasks
02 Initialise ReadyList
02 Repeat
03      Select ready task with highest priority
04      (Assign task to a processing node)
05      Schedule task (Set its start time)
06      Update ReadyList
07 Until ReadyList is empty
```

Figure 3.6: Static Cyclic Scheduling
The List Scheduling Algorithm

3.5 Static Cyclic Scheduling of Time-Triggered Activities in a Heterogeneous TT/ET Environment

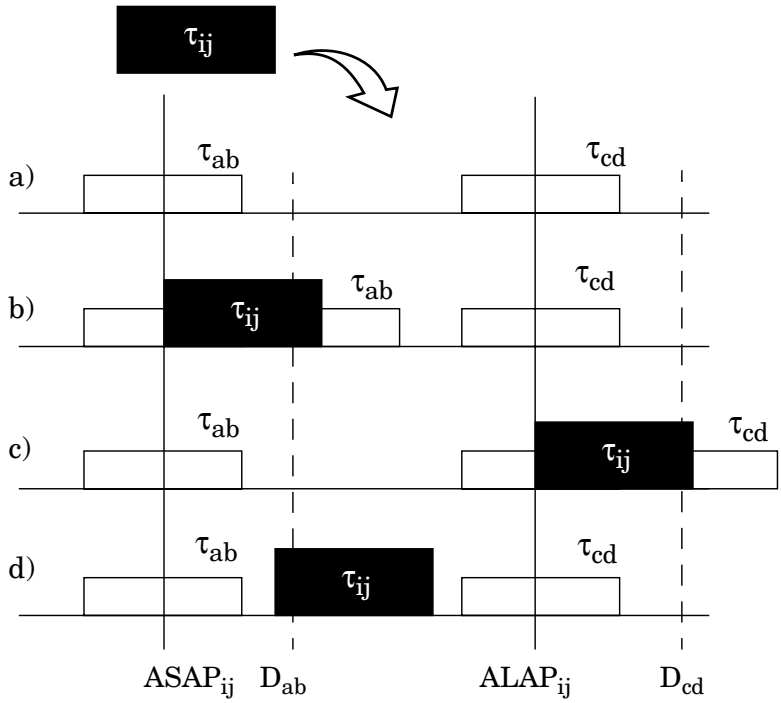
For the construction of the static cyclic schedule for TT tasks and ST messages, we use a list scheduling based algorithm [Coff72]. Assuming that in our application we have N time-triggered task graphs Γ_1, Γ_2, ..., Γ_N, the static schedule will be computed over a period T_{SS} = LCM(T_1, T_2, ..., T_N). The input to the list scheduling algorithm is a graph consisting of n_i instances of each Γ_i, where $n_i = T_{SS} / T_i$.

A generic list scheduling algorithm is illustrated in Figure 3.6. The algorithm starts by associating priorities to each of the tasks in the task set. Then, in line 02, a ReadyList is created with all the tasks which have no predecessors and therefore are ready for being scheduled. The body of the algorithm consists of a loop (lines 02-07) which, during each iteration, selects from ReadyList the task with the highest priority, assigns it to a processor if it was not already mapped, and then schedules the task. Before a new iteration starts, the ReadyList is updated by adding those tasks which have all their predecessors scheduled. The algorithm ends when all the tasks have been scheduled and the ReadyList is empty. Alternative versions of list scheduling differ

from each other mainly in the way the task priorities are computed (for example, the priorities can be updated/recomputed after each loop) and the way tasks are scheduled (like ASAP - as soon as possible, for example).

Considering our particular problem formulation, the generic list scheduling algorithm presented in Figure 3.6 needed several modifications. First, the ReadyList contains all TT tasks *and* ST messages which are ready to be scheduled (they have no predecessors or all their predecessors have been scheduled). From the ready list, tasks and messages are extracted one by one to be scheduled on the processor they are mapped to, or into a static bus-slot associated to that processor on which the sender of the message is executed, respectively. Second, the priority function which is used to select among ready tasks and messages is a critical path metric, modified for the particular goal of scheduling tasks mapped on distributed systems [Pop00].

However, an important aspect of the static scheduling for heterogeneous TT/ET systems is represented by the actual scheduling of a given TT task, i.e. determining its start time such that the interference on ET activities is minimised (line 05 in Figure 3.6). For example, let us consider a particular task τ_{ij} selected from the ready list to be scheduled. We consider that $ASAP_{ij}$ is the earliest time moment which satisfies the condition that all preceding activities (tasks or messages) of τ_{ij} in graph Γ_i are finished and $Node_{ij}$ is free. The moment $ALAP_{ij}$ is the latest time when τ_{ij} can be scheduled. With only the TT tasks in the system, the straightforward solution would be to schedule τ_{ij} at $ASAP_{ij}$. In the example illustrated in Figure 3.7.b), however, such a solution could have negative effects on the schedulability of the ET task τ_{ab}, which misses its deadline D_{ab}. In the case when task τ_{ij} is scheduled as late as possible (Figure 3.7.c), task τ_{cd} misses its deadline D_{cd}; moreover, because of data dependencies, there is a risk that, by scheduling τ_{ij} at $ALAP_{ij}$, the TT tasks which are not scheduled yet will miss their deadline (this aspect will be discussed more in the following sub-section). What we have to do is



a) τ_{ij} not scheduled yet
b) τ_{ij} scheduled ASAP, τ_{ab} misses its deadline
c) τ_{ij} scheduled ALAP, τ_{cd} misses its deadline
d) τ_{ij} scheduled inside [$ASAP_{ij}$, $ALAP_{ij}$] so that tasks τ_{ab} and τ_{cd} are not disturbed and meet their deadlines

Figure 3.7: Scheduling a TT task in a mixed TT/ET environment

to place task τ_{ij} in such a position inside the interval [$ASAP_{ij}$, $ALAP_{ij}$] so that the chance to finally get a globally schedulable system is maximised.

The number of possible scheduling times for task τ_{ij} in the interval [$ASAP_{ij}$, $ALAP_{ij}$] is potentially infinite. In order to con-

sider only a limited number of possible alternatives for the start time of a TT task τ_{ij}, we take into account the information obtained from the schedulability analysis described in Section 3.3, which allows us to compute the response times of ET tasks. We started from the obvious observation that statically scheduling τ_{ij} after an ET task τ_{kl} has finished its execution will guarantee that task τ_{ij} will not interfere with τ_{kl}. Thus, we consider as alternative start times for τ_{ij} the response times of all ET tasks which finish their execution inside the $[ASAP_{ij}, ALAP_{ij}]$ interval:

$$\text{alternative\_start\_times}(\tau_{ij}) = \{ASAP_{ij}\} \cup ([ASAP_{ij}, ALAP_{ij}] \cap \{R_{kl} \mid \tau_{kl} \in ETdomain, Node_{kl} = Node_{ij}\})$$

The moment referred by $ASAP_{ij}$ is added to *alternative_start_times* so that the set of alternative start times of a TT task will not be empty even if no ET tasks finish their execution during the interval $[ASAP_{ij}, ALAP_{ij}]$.

We illustrate the choice of possible start times of a TT task τ_{ij} in Figure 3.8 where three ET tasks τ_{kl}, τ_{kl+1}, τ_{kl+2} finish their execution inside $[ASAP_{ij}, ALAP_{ij}]$ leading to alternative_start_times(τ_{ij}) = $\{ASAP_{ij}, R_{kl}, R_{kl+1}, R_{kl+2}\}$. Statically scheduling τ_{ij} at time R_{kl} avoids the interferences from τ_{ij} to τ_{kl}, while scheduling τ_{ij} even later, at R_{kl+1}, will guarantee that τ_{ij} does not interfere with either τ_{kl} or τ_{kl+1}.

After identifying the set of candidate start times of a task, we have to select one of them as the static schedule for that task. Two aspects have to be considered in this context:

1. The interference with the ET activities should be minimised;
2. The deadlines of TT activities should be satisfied.

In order to evaluate the first goal, the value of the function *DSch* (see Section 3.3) is computed for each alternative start time t after performing the schedulability analysis of the ET task set considering the influence from the TT tasks, with τ_{ij} scheduled at t. As will be shown in the following section, a global

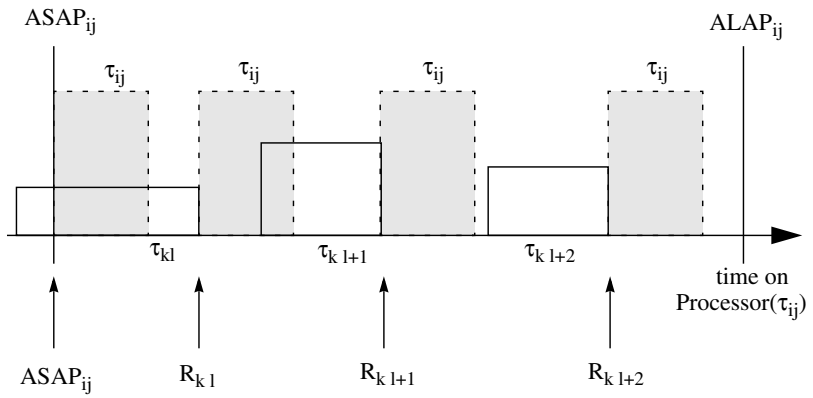


Figure 3.8: Selection of Alternative Start Times

cost function is computed, which combines both goals defined above, and, based on a greedy approach, the start time of the task will be selected.

The scheduling algorithm is presented in Figure 3.9. If the selected TT activity extracted from the *ready_list* is a task τ_{ij}, then the *alternative_start_times* are evaluated and the algorithm selects the one which generates the smallest value of the cost function. When scheduling an ST message extracted from the ready list, we place it into the first bus-slot associated with the sender node in which there is sufficient space available. If all TT tasks and ST messages have been scheduled and the schedulability analysis for the ET tasks indicates $DSch \leq 0$, then the global system scheduling has succeeded.

For the case that no correct schedule has been produced, we have implemented a backtracking mechanism in the list scheduling algorithm, which allows to turn back to previous scheduling steps and to try alternative solutions. In order to avoid excessive scheduling times, the maximum number of backtracking steps can be limited.

```
00 compute [ASAP,ALAP] for each TT activity
01    while ready_list is not empty
02        select TT activity τij
03        if τij is a task
04            Schedulability Analysis -> Compute
                  response times of ET activities
05            Compute alternative_start_times(τij)
06            for t in alternative_start_times(τij
07                schedule τij at t
08                Schedulability Analysis ->
                      Compute DSch ->
                      Compute CostFunction
09            endfor
10            schedule τij at t for which
                  the CostFunction is minimum
11        else // τij is a message
12            ASAP schedule τij in slot_i
13        endif
14        update ready_list
15     endwhile
```

Figure 3.9: Static Scheduling Algorithm

In the following subsections we present three alternative ways to compute the cost function which drives the heuristic in Figure 3.9. The three alternatives are identified as MxS1, MxS2 and MxS3 (from mixed scheduling).

3.5.1 MXS1

Scheduling a TT task τ_{ij} inside its $[ASAP_{ij}, ALAP_{ij}]$ interval will, of course, guarantee that deadlines related to this particular task are satisfied, and that there exists the possibility that a valid static schedule can be constructed for the system. However, due to the data dependencies, scheduling τ_{ij} later inside $[ASAP_{ij}, ALAP_{ij}]$ decreases the probability of finding a feasible static schedule for the tasks further down. This is why, for the

evaluation of the alternative start times of a TT task τ_{ij} (line 08 in Figure 3.9), we introduced a cost function which combines the degree of schedulability of the ET activities (*DSch* in Section 3.3) with a second metric which captures the "risks" taken by scheduling τ_{ij} at later times:

$$f(t, \tau_{ij}) = A \cdot e^{-slack(t, \tau_{ij})} \cdot t + B \cdot DSch$$

where t is one of the alternative start times, A and B are normalisation constants, and *slack(t,* τ_{ij}*)* represents the available processing capability on $Node_{ij}$ (the processing time inside the interval $[t + C_{ij}, T_{SS}]$ which is not used by any of the ET or TT tasks). The value of *slack* is computed as follows:

$$slack(t, \tau_{ij}) = [T_{SS} - (t + C_{ij})] - H_{ET}(t + C_{ij}, T_{SS}, Proc_{ij}) - UnschH_{TT}(Proc_{ij})$$

where $UnschH_{TT}$ represents the sum of execution times of all yet unscheduled TT tasks mapped on $Node_{ij}$. The term H_{ET} represents the time demanded by ET tasks in the interval $[t + C_{ij}, T_{SS}]$ on $Node_{ij}$ and is computed in the following steps:

1. For each ET task τ_{ab} mapped on $Node_{ij}$ consider its worst case response interval $I_{ab} = [\phi_{ab}, R_{ab}]$ using the response times computed in line 08 of the algorithm in Figure 3.9.
2. For each scheduled TT task τ_{ab} mapped on $Node_{ij,}$ we know the start time t_{ab} and consider the associated execution interval $I_{ab} = [t_{ab}, t_{ab} + C_{ab}]$.
3. Compute the unions of intervals in which ET and scheduled TT activities take place: $I_{ET} = \bigcup_{\forall \tau_{ab} \in ET} I_{ab}$ and $I_{schedTT} = \bigcup_{\forall \tau_{ab} \in schedTT} I_{ab}$.
4. Compute H_{ET} as the sum of lengths of each of the intervals in $(I_{ET} \cup I_{schedTT}) \cap [t_{ij} + C_{ij}, T_{SS}]$.

It is easy to notice that if the *slack* has a very small value (even

negative, if the demand is higher than the available processing time), then the first term in function f (the one depending on time t) has a much greater weight on the value of f than the second term of the function f. Consequently, earlier start times for τ_{ab} will be preferred. On the other hand, if there is more available processor time than needed (in other words, *slack* has a high value), the function f will depend mainly on the value of the second term, thus the main aspect taken into consideration will be the influence of TT activities on the ET ones, which is captured by $DSch$.

The static scheduling algorithm will select, among the alternative start times, that time t for which the value of the cost function f is minimum.

3.5.2 MxS2

The schedulability analysis algorithm described in Section 3.3 is applied very often during the static scheduling procedure presented in Figure 3.9, both in order to compute the values of the possible start times (line 04) and the *Cost* associated with each such start time (line 08). In order to reduce the amount of time needed for scheduling, we experimented with an algorithm which uses the schedulability analysis only for determining the set of *start_times*(line 04), while the evaluation process in line 08 is based on a simpler version of function f. In MxS1, when the alternative start times of a TT task are evaluated, running the schedulability analysis returns the value $DSch$ which reflects the amount of new interference that has been introduced in the ET subsystem at a global level. The simpler function f, which we use in this second algorithm, avoids calling the global schedulability analysis for each possible start time of a TT task τ_{ij} and considers *only* the interferences produced by τ_{ij} on the ET tasks mapped on $Node_{ij}$.

$$f'(t, \tau_{ij}) = A \cdot e^{-slack(t, \tau_{ij})} \cdot t + B \cdot (DSch + \Delta DSch),$$

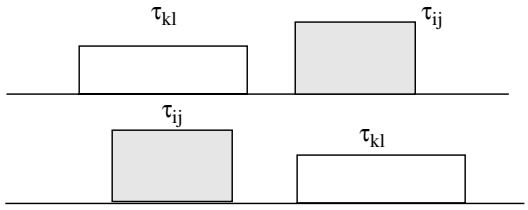
where the value of *DSch* (as expressed in Section 3.3) is computed (on line 4, Figure 3.9) before τ_{ij} has been scheduled, and $\Delta DSch$ is the amount of interference introduced by τ_{ij} on the ET tasks mapped on $Node_{ij}$:

$$DSch = \sum_{\substack{\tau_{kl} \in ET \\ Node_{kl} = Node_{ij}}} \left(R'_{kl} - R_{kl} \right)$$

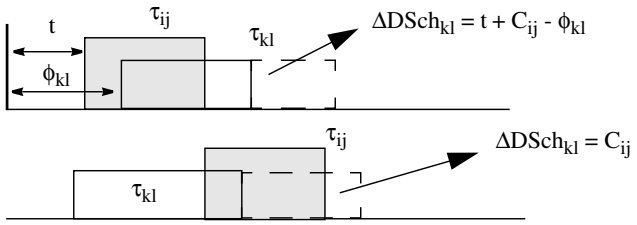
where R_{kl} is the response time of an ET task τ_{kl} before τ_{ij} has been scheduled and R'_{kl} is an approximation of the response time of τ_{kl} after τ_{ij} has been scheduled at time t. We estimate that, depending on the time t when a TT task τ_{ij} is scheduled, the response time of an ET task τ_{kl} mapped on the same $Node_{ij}$ either remains unchanged (is not influenced at all) or is increased with a value up to the worst case execution time C_{ij} of the TT task.

Figure 3.10 presents the situations when the response time of an ET task τ_{kl} remains unchanged (Figure 3.10.a) and when it is increased because of the influence of a TT task τ_{ij} (Figure 3.10.b). The cases represented in Figure 3.10.a) show that when a TT task τ_{ij} is scheduled at time t so that its associated execution interval $[t, t + C_{ij}]$ does not intersect with the time interval where an ET task executes in the worst case $[\phi_{kl}, R_{kl}]$, then we estimate that after scheduling τ_{ij} at t, the response time for τ_{kl} will be the same, $R'_{kl} = R_{kl}$. However, if the intersection is not empty (like in the cases in Figure 3.10.b), then $R'_{kl} = R_{kl} + \Delta DSch_{kl}$. The value for the increment used in the function $f'(t,\tau_{ij})$ will be computed as $\Delta DSch = \Sigma \Delta DSch_{kl}$, for all τ_{kl} in the ET domain and $Node_{kl} = Node_{ij}$.

In this algorithm, MxS2, the schedulability analysis of the system is called only once for each TT task (step 04), which will lead, as we will see in Section 3.6, to shorter computation times.



a) R_{kl} remains unchanged



b) R_{kl} is increased with ΔDSch

Figure 3.10: Estimation of ET response times in MxS2

3.5.3 MXS3

List scheduling, which is the basis for our scheduling algorithm, is a constructive method that builds the static schedule table incrementally, by adding one TT task or ST message at a time. In the previous two versions of the algorithm (Section 3.5.1 and Section 3.5.2), at each step, the effect of the static schedule, including the newly introduced task, on the ET subsystem is measured by function *DSch*. However, the problem is that the available static schedule is not complete when estimating, for a given task τ_{ij}, the global influence of TT activities on the set of ET ones. For the alternatives MxS1 and MxS2 we have chosen the following simple approach: for evaluating the influence of the decision of which alternative start time to select for τ_{ij}, we

consider only that part of the static schedule which already has been built, up to that particular moment. The selection is fair, as the same conditions are applied to all alternative times; however, it is inaccurate, since a part of the final static schedule is ignored when taking the decision. For the alternative MxS3 we have considered a solution which tries to improve on this lack of accuracy by considering the whole set of TT activities when evaluating the degree of schedulability of the ET tasks and messages. This is solved by considering an approximate static schedule for the yet unscheduled TT activities. Therefore, a preliminary step is performed in preparation of the algorithm in Figure 3.9.

First, we build an initial static schedule by using a simpler and faster version of the algorithm in Figure 3.9. In this version, the response times of the ET activities are computed only once in the beginning of the algorithm and the evaluation of possible start times is performed using a simple function like in MxS2. This step allows us to rapidly obtain a static schedule which will be at the basis of the second step of our approach.

After the preparation step, we run the algorithm in Figure 3.9, but whenever schedulability analysis of the ET subsystem is performed, we consider that the interfering static schedule not only contains the TT activities which were scheduled so far, but all the TT tasks and ST messages in the system. We obtain such a complete static schedule by considering:

- the start times of the TT tasks/ ST messages scheduled so far in this second step;
- for the unscheduled TT tasks/ ST messages, we consider their start times as identified in the first step of the algorithm

Figure 3.11 illustrates the way we obtain such a complete static schedule. The static schedule considered during the schedulability analysis of the ET subsystem (Figure 3.11.c) contains all the TT tasks and ST messages in the system. Such a com-

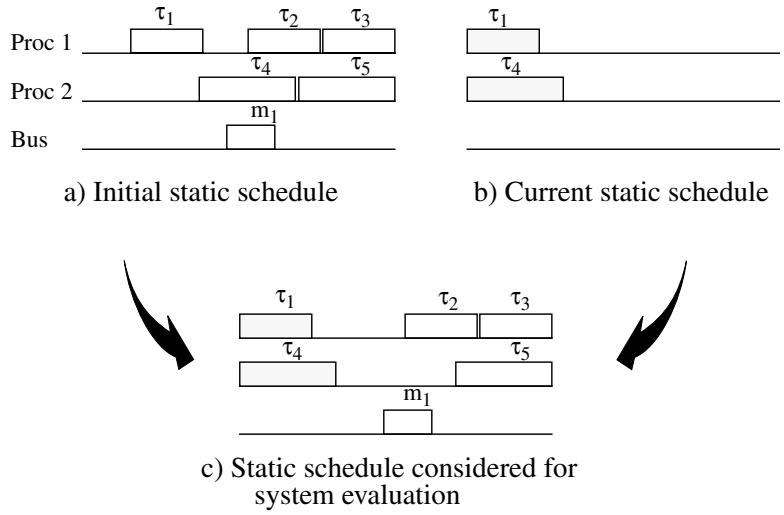
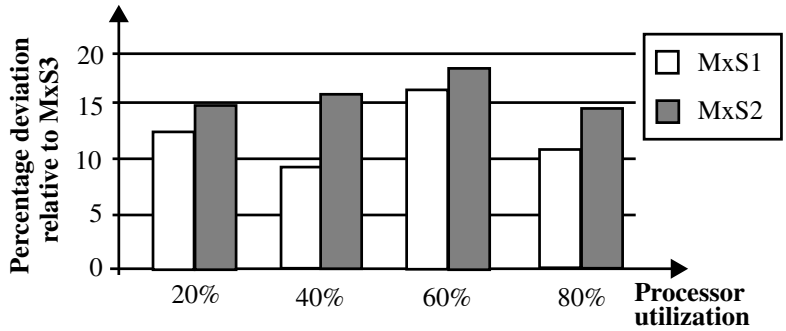


Figure 3.11: Construction of a static schedule for complete evaluation of the system timing properties

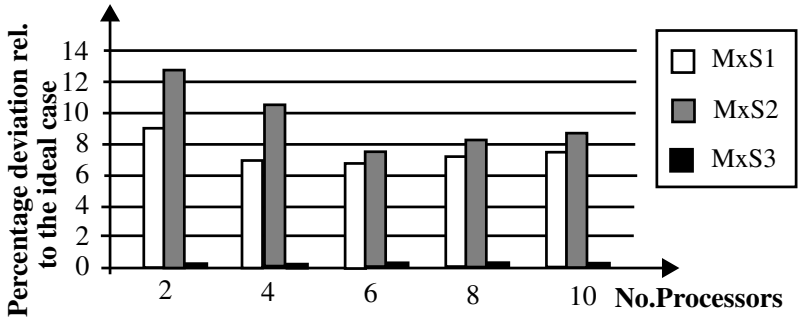
plete schedule is obtained by putting together the start times for tasks τ_1 and τ_4 (which have been scheduled already, see Figure 3.11.b) and the start times identified in the first step for the unscheduled tasks τ_2, τ_3, τ_5 and message m_1 (Figure 3.11.a).

3.6 Experimental Results

For evaluation of our scheduling and analysis algorithm we generated a set of 3600 tests representing systems of 2 to 10 nodes. The number of tasks mapped on each node varied between 10 and 30, leading to applications with a number of 20 up to 300 tasks. The tasks were grouped in task-graphs of 5, 10 or 15 tasks. Between 20% and 80% of the total number of tasks were considered as event-triggered and the rest were set as time-triggered. The execution times of the tasks were generated in such a way that the utilisation on each processor was between 20% and 80%. In a similar manner we assured that 20% and up to 60% of



a) Deviation of Number of solutions (smaller is better)



b) Degree of interference (smaller is better)

Figure 3.12: Evaluation of Scheduling Heuristics MxS1, MxS2 and MxS3

the total utilisation on a processor is required by the ET activities. All experiments were run on an AMD Athlon 850MHz PC.

The first set of experiments compares the three versions of the holistic scheduling algorithm we proposed in Section 3.5. In Figure 3.12.a) we illustrate the capacity of MxS1 and MxS2 to produce schedulable systems, compared to that of MxS3. For example, in the case of a 60% load, MxS2 was able to generate 18% and MxS1 16% less schedulable solutions than MxS3. In addition, for each heuristic, we computed the quality of the identified solutions, as the percentage deviation of the schedulability

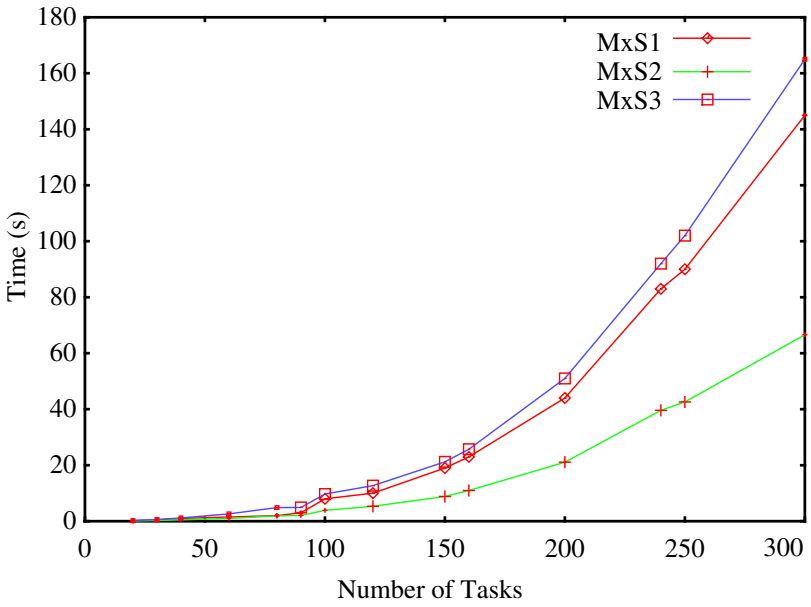


Figure 3.13: Average Computation Times

degree ($DSch_{MxS}$) of the ET activities in the resulted system, relative to the schedulability degree of an ideal solution ($DSch_{ref}$) in which the static schedule does not interfere at all with the execution of the ET activities:

$$Interference = \frac{DSch_{ref} - DSch_{MxS}}{DSch_{ref}} \cdot 100$$

In other words, we used the function $DSch$ as a measure of the interference introduced by the TT activities on the execution of ET activities. In Figure 3.12.b), we present the average quality of the solutions found by the three algorithms. For this diagram, we used only those results where all three algorithms managed to find a schedulable solution. It is easy to observe that the solutions obtained with MxS3 are constantly at a minimal level of interference. The heuristics MxS1 and MxS2 produce solutions in which the TT interference is considerably higher, resulting in significantly larger response times of the ET activities and con-

sequently to a decrease of the schedulability degree by 7-13%. Not surprisingly, our experiments prove that the heuristic MxS3 is the most accurate and consequently produces results of the best quality. MxS2, which uses local approximation for the evaluation of the ET schedulability, has a slightly lower quality than MxS1.

In Figure 3.13 we present the average execution times of the three scheduling heuristics. According to expectations, MxS2 is the fastest of the three heuristics, while MxS3 is slightly slower than MxS1. In conclusion, the heuristic MxS3 is the one which offers the best solutions at an acceptable computation time. MxS2 is very fast and can be used in certain particular cases like, for example, inside a design space exploration loop with an extremely large number of iterations.

Chapter 4
Design Optimisation Of Heterogeneous Time/Event-Triggered Systems

NOW THAT WE ARE ABLE to derive the schedulability degree of a heterogeneous TT/ET system, we consider in this chapter a larger design context, which involves mapping, scheduling and a couple of specific optimisation aspects which are characteristic for this type of systems.

4.1 Specific Optimisation Problems

4.1.1 PARTITIONING OF SYSTEM FUNCTIONALITY INTO EVENT AND TIME-TRIGGERED ACTIVITIES

During the design process, a decision should be made on which tasks and messages will be implemented as TT/ET and ST/DYN activities, respectively. Typically, this decision is taken, based on the experience and preferences of the designer considering aspects like the functionality implemented by the task, the

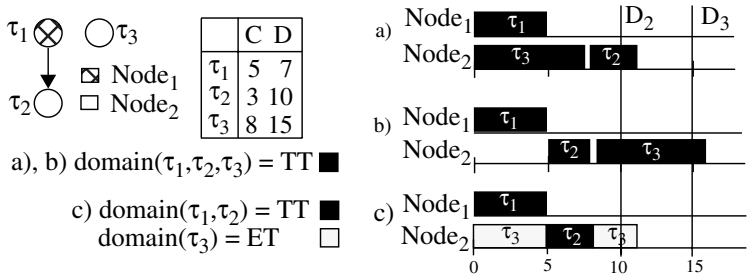


Figure 4.1: Partitioning into TT/ET domains (TT→ET)

hardness of the constraints, sensitivity to jitter, legacy, etc. There exists, however, a subset of tasks/messages which could be assigned to any of the domains. Decisions concerning the partitioning of this set of activities can lead to various trade-offs concerning, for example, the size of the schedule table or the schedulability properties of the system. For example, in Figure 4.1 we show a system with two nodes on which three tasks are mapped: τ_1 on $Node_1$, τ_2 and τ_3 on $Node_2$; τ_2 is data dependant on τ_1; worst case execution times (C_i) and deadlines (D_i) are shown in the figure. In order to keep the example simple, communication delays are ignored. When all three tasks belong to the TT domain, the system is unschedulable. In this case, either τ_2 (scheduling alternative depicted in Figure 4.1.a) or τ_3 (Figure 4.1.b) misses its deadline. If, however, τ_3 is moved into the ET domain (Figure 4.1.c), all tasks are schedulable (in this case, τ_2 will pre-empt the execution of τ_3).

In Figure 4.2 we illustrate the opposite movement, from the ET into the TT domain. The ET task τ_1 cannot meet its deadline D_1, because no matter how much its priority is increased, it will still be pre-empted and delayed by the execution of a TT task τ_2

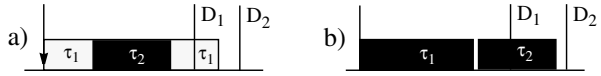


Figure 4.2: Partitioning into TT/ET domains (ET→TT)

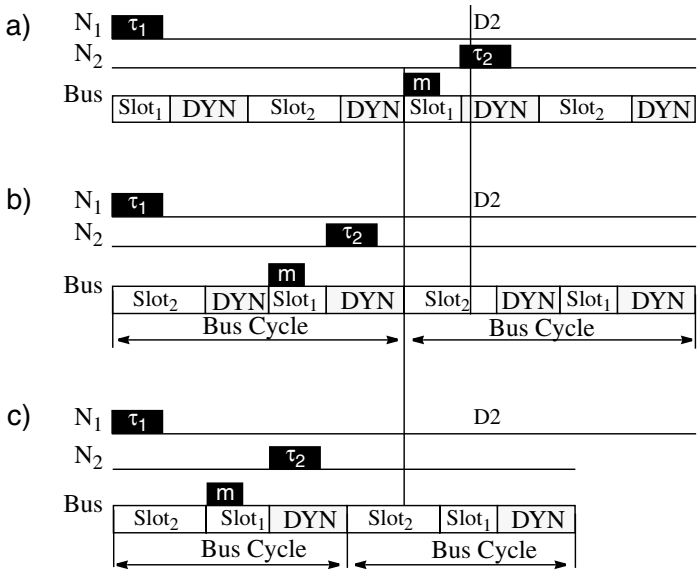


Figure 4.3: Optimization of Bus Access Cycle

which is mapped on the same node (Figure 4.2.a). In Figure 4.2.b), task τ_1 has been assigned to the TT domain and it meets its deadline because the static scheduling algorithm will be able to avoid the previous resource conflict.

4.1.2 BUS ACCESS OPTIMISATION

The configuration of the bus access cycle has a strong impact on the global performance of the system. The parameters of this cycle have to be optimised such that they fit the particular application and the timing requirements. Parameters to be optimised are the number of static and dynamic phases during a communication cycle, as well as the length and order of these phases. Considering the static phases, parameters to be fixed are the order, number, and length of slots assigned to the different nodes. For example, consider the situation in Figure 4.3, where task τ_1 is mapped on node N_1 and sends a message m to task τ_2 which is mapped on node N_2. In case a), task τ_1 misses the start

of the ST $Slot_1$ and, therefore, message m will be sent during the next bus cycle, causing the receiver task τ_2 to miss its deadline D_2. In case b), the order of ST slots inside the bus cycle is changed, the message m will be transmitted earlier and τ_2 will meet its deadline. The resulted situation can be further improved, as it can be seen in Figure 5.c), where task τ_2 finishes even earlier, if the first DYN phase in the bus cycle can be eliminated without producing intolerable delays of the DYN messages (which have been ignored in this example).

4.2 Problem Formulation

We consider a system specification and an architecture as described in Chapter 2. We also consider that some of the tasks are already mapped to nodes and their domain (TT or ET) is fixed. This can be the result of decisions already taken by the designer or/and because part of the functionality is inherited from previous generations of the product. However, we assume that there are tasks which are not mapped yet and some of the task graphs are not yet partitioned between the two domains. We denote with Ψ^{P} the set of all tasks which are not yet assigned to any of the ET or TT domains and with Ψ^{M} the set of all tasks which are not mapped to any node. Note that $\Psi^{\mathrm{P}} \cap \Psi^{\mathrm{M}}$ is not necessarily empty, which means that some tasks have neither a fixed domain, nor are they mapped on any node. The tasks in the set $\Psi^{\mathrm{P}} \cup \Psi^{\mathrm{M}}$ are those to which we refer in the rest of this chapter when we discuss mapping and partitioning. None of the other tasks is affected, in terms of partitioning and mapping, by any of the design decisions. Our goal is threefold:

- to partition the task set Ψ^{P} among the ET and TT domains;
- to map the tasks in the set Ψ^{M} onto the nodes in the architecture; and
- to optimise the parameters of the communication protocol.

The above design tasks have to be performed with the overall goal that the timing constraints of the resulted system are satis-

fied. If this is achieved, we say that we have obtained a schedulable implementation of the system, which implies that the following two conditions are satisfied:

1. The tasks in the TT domain are schedulable, meaning that we were able to build a static schedule (Section 3.5) for all the tasks in the TT partition such that their deadlines are satisfied;
2. The tasks in the ET domain are schedulable. This means that after running the scheduling process described in Section 3.5, the function $DSch$, expressing the schedulability degree of the ET activities, will have a value $DSch \leq 0$.

Before starting to discuss the actual heuristics, some further observations have to be made. According to the application model presented in Section 2.4, all tasks in a task graph belong to the same domain. Thus, the task set Ψ^P contains complete task graphs and, by deciding on the partitioning of a certain task, the whole task graph is assigned to either the TT or ET domain.

A similar partitioning problem, as formulated above for tasks, could be also defined at the level of messages: considering a set of messages, for each message it has to be decided if it should be transmitted in an ST phase (statically scheduled) or in a DYN phase (dynamically scheduled). In order to keep the presentation reasonably simple, we consider that all messages are preassigned as ST or DYN. For the same reason, we also consider that all tasks in the set Ψ^P have a pre-assigned priority which is used if the task is assigned to the ET domain.

4.3 Design Heuristic

The design problem outlined above is a combination of subproblems, each of exponential complexity. Therefore, we have elaborated a design space exploration strategy based on the application of several heuristics in three successive steps, as

```
01 Gen_Part, Gen_Map, Gen_Bus_Cycle
02 if TT not schedulable then
03    change partitioning (TT to ET)
04    change mapping
05    change bus cycle
06 endif
07 if TT not schedulable then stop endif
08 if ET not schedulable then
09    Mapping_and_Partitioning
10    if ET not schedulable then
11        Optimize_Bus_Access
12    endif
13 endif
```

Figure 4.4: Overview of the Design Heuristic

shown in Figure 4.4:

1. The first step (lines 01-06) starts by generating an initial mapping, partitioning and bus structure, using several basic criteria (line 01). If this initial solution is not schedulable, successive transformations are applied to the partitioning, mapping, and the bus cycle, with the aim of finding a solution such that the TT tasks are schedulable. This is performed by generating configurations (in terms of partitioning, mapping and bus cycle) which are more and more favourable to the TT partition.
   The first step is stopped once a solution with a schedulable TT partition has been reached. If at the end of the first step no such solution has been found, we conclude that, given the amount of available resources, no correct implementation of the system can be generated. This decision is justified by the fact that, if under the most favorable conditions no static schedule could be generated for the TT tasks, no further design transformations could lead to a globally schedulable solution, except for modifications of the underlying system architecture (e.g adding a new node, replacing a node with a

faster one or a similar replacement of the bus). If the configuration generated after the first step is not globally schedulable, but a correct schedule of the TT tasks and ST messages has been found, the heuristic moves into the second step.

2. During the second step (line 09), a partitioning and mapping algorithm tries to produce a solution such that not only the TT static schedule is correct, but also the degree of schedulability of the ET partition is as good as possible. The cost function driving the design space exploration during this step is *DSch* (see Section 3.3). Simultaneously with each partitioning and mapping decision, also the bus cycle is modified in order to fit the new configuration.
3. If the second step did not succeed in producing a schedulable ET partition, the third step (line 11) tries to further improve the degree of schedulability by an aggressive optimisation of the bus access cycle.

In the following subsections we further elaborate on the optimisation steps outlined above.

4.3.1 BUILDING AN INITIAL CONFIGURATION

The first step starts with generating, based on a very simple and fast heuristic, a mapping and partitioning of the tasks, as well as a bus cycle (line 01 in Figure 4.4):

- The partitioning is performed with the only constraint to evenly distribute the load between the TT and the ET domains.
- The mapping is based on a very fast heuristic aimed at minimising inter-processor communication while keeping a balanced processor load.
- The initial bus cycle is constructed in the following two steps:

  1. We consider that each node can transmit messages during only one ST slot inside a bus cycle. The ST slots are assigned

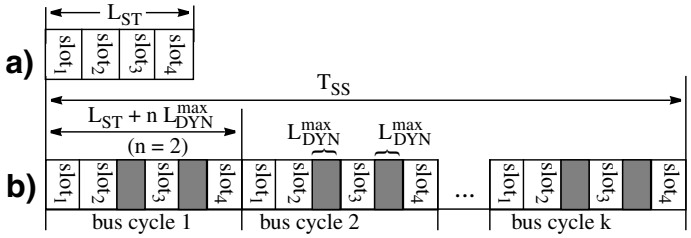


Figure 4.5: Initial Bus Configuration

in order to the nodes such that $Node_i$ transmits during $Slot_i$ (Figure 2.1). The length of $Slot_i$ is set to a value which is equal to the length of the largest ST message generated by a task mapped on $Node_i$. Considering an architecture of 4 nodes, a structure like the one in Figure 4.5.(a) is produced after this step.

2. Dynamic phases are introduced in order to generate a mixed ST/DYN bus cycle. We start from the rough assumption that the total length of the dynamic phases over a period T_{SS} (T_{SS} is the length of the static schedule, see Section 3.5) is equal to the total length of the DYN messages transmitted over the same period, which is:

$$\sum_{m_i \in DYNdomain} \frac{T_{SS}}{T_i} \cdot L_i,$$

where T_i and L_i are the period and the length (expressed in time units) of the DYN message m_i. We set the length of each DYN phase to the length L_{DYN}^{max} of the largest DYN message. The number n of dynamic phases in each cycle can be determined from the following equation:

$$\frac{T_{SS}}{L_{ST} + n \cdot L_{DYN}^{max}} \cdot n \cdot L_{DYN}^{max} = \sum_{m_i \in DYNdomain} \frac{T_{SS}}{T_i} \cdot L_i,$$

where L_{ST} is the total length of the static slots in a bus cycle and $L_{ST} + n\ L_{DYN}^{max}$ is the length of the bus cycle. Finally, the dynamic phases are evenly distributed inside the bus cycle. Figure 4.5.(b) illustrates such an initial bus configuration.

4.3.2 ADJUSTING THE INITIAL CONFIGURATION

Once we have decided on the above configuration, we can run the holistic scheduling algorithm, which will lead to one of the following outcomes:

a. the system is found schedulable;

b. the TT activities are schedulable but the ET ones are not, i.e. a valid static schedule has been built but the analysis has identified at least one ET activity for which $R_{ij} > D_{ij}$;

c. the ET activities are schedulable but the TT ones are not;

d. both ET and TT activities are not schedulable.

In the first case, a), the design goal has been achieved and, therefore, no further optimisations are performed. In the cases c) and d), we perform the following successive operations, aimed at achieving a schedulable TT domain (lines 03-05 in Figure 4.4):

1. Task graphs are moved, one by one, from the TT to the ET domain, until either the remaining TT activities are schedulable or there are no more task graphs to be moved (whole task graphs are moved and not individual tasks, because, as mentioned earlier, all tasks in a task graph belong to the same domain). The order in which task graphs are moved is based on a priority function that captures the mobility of tasks in the graph:

$$RelAvgMob(\Gamma_i) = \frac{1}{n_i} \cdot \sum_{j=1}^{n_i} \frac{D_{ij} - ASAP_{ij}}{C_{ij}}$$

where n_i is the number of tasks in the task graph Γ_i, and $ASAP_{ij}$ is the earliest possible start time for task τ_{ij}. Task graphs with a low average relative mobility are moved first, because, in principle, they are the most difficult to be scheduled statically.

2. If the TT domain still is unschedulable, TT tasks are remapped with the goal of avoiding unbalanced node utilisation by TT tasks.
3. If no schedulable TT domain has been yet produced, transformations of the bus cycles are performed such that the delays produced by ST messages are reduced. In this step, a simpler and faster version of the heuristic presented in Section 4.3.4 is used.

If no schedulable TT domain has been produced by the above transformations, no correct implementation can be obtained with our heuristic given the available resources. If both the TT and ET domains are schedulable, we have achieved our design goal, while in the case of an unschedulable ET domain, the heuristic is continued with the second step.

4.3.3 MAPPING, PARTITIONING AND SCHEDULING

The mapping and partitioning step (line 09 in Figure 4.4) receives as an input a configuration in which the TT activities are schedulable and the ET ones are not. The algorithm is illustrated in Figure 4.6. It selects iteratively tasks $\tau_{ij} \in \Psi^P \cup \Psi^M$ (line 03) in order to be remapped and/or repartitioned. The order in which tasks are processed is defined by the following two rules, similar to those used in list scheduling:

1. τ_{ij} is selected only after all its predecessors in the task graph Γ_i have already been processed (these tasks are called *ready*).
2. Among the ready tasks, the selection is based on a priority function PF similar to the one proposed in [Pop00] (line 03).

```
01 while (Ψ^P ∪ Ψ^M ≠ ∅ and BestCost > 0) do
02    update priority function PF
03    select task τ_ij ∈ Ψ^P ∪ Ψ^M with highest PF
04    BestCost = ∞
05    if τ_ij ∈ Ψ^M then          -- task is not mapped
06        for (p = 1 to NrNodes) do
07            map τ_ij on Node_p
08            adjust bus access cycle
09            if τ_ij ∈ Ψ^P then Cost,d = partition(τ_ij)
10            else Cost = DSch; d = domain of τ_ij
11            endif
12            if BestCost > Cost then
13                BestCost = Cost; BestDomain = d;
14                BestNode = p; BestCycle = BusCycle;
15            endif
16        endfor
17    else
18        BestNode = Node on which τ_ij is mapped
19        BestCycle = BusCycle;
20        BestCost, BestDomain = partition(τ_ij);
21    endif
22    τ_ij.node = BestNode; BusCycle=BestCycle;
23    set domain(Γ_i) to BestDomain
24    Ψ^P=Ψ^P \ {τ_ij}; Ψ^M = Ψ^M \ {τ_ij}
25 end while

26 function partition(τ_ij)
27    τ_ij.domain = ET; Cost1 = DSch; d_1 = ET;
28    τ_ij.domain = TT; Cost2 = DSch; d_2 = ET;
29    return min(Cost1, Cost2) and associated d_i
30 end partition
```

Figure 4.6: Mapping and Partitioning Algorithm

This function is based on a critical path metric and it also takes into consideration the delay introduced by message passing considering the particular communication protocol, as well as the nature of the messages (ST or DYN).

Once a task τ_{ij} has been selected, its mapping and domain will be decided in a greedy fashion. If $\tau_{ij} \in \Psi^{M}$ (the task mapping is not fixed), it will be successively mapped to each node (lines 06-16) and for each alternative, the schedulability analysis (Section 3.3) returns the cost *DSch*, which captures the degree of schedulability of the produced configuration. If the domain, ET or TT, is also to be decided ($\tau_{ij} \in \Psi^{P}$), both alternatives are evaluated (line 09). This is performed using the function `partition` (lines 26-30). Finally, that node and domain are selected for τ_{ij} which produces the smallest value for *DSch*. If only the domain of τ_{ij} is to be decided, but the mapping is fixed, the best of the two alternatives is selected. It should be mentioned that a mapping or partitioning alternative is considered only if, with the resulted configuration, the TT domain is still schedulable (this aspect is not captured in Figure 4.6).

Whenever the mapping of a task is modified, the bus cycle has to be adjusted so that it can ensure the minimum requirements for transmitting the messages. Such an adjustment of the bus access cycle is illustrated in Figure 4.7, where 4 TT tasks are mapped on 3 nodes (N_1, N_2 and N_3). The number at the side of

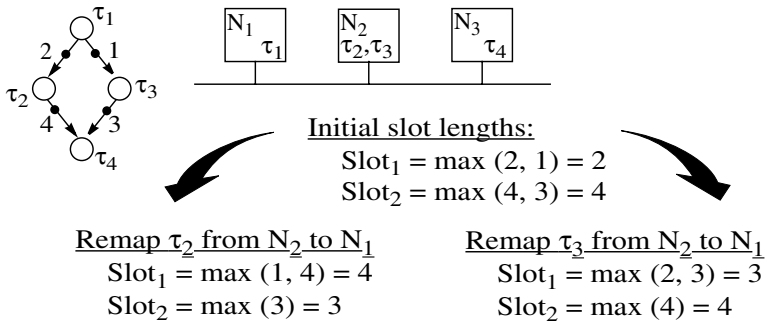


Figure 4.7: Adjustment of the Bus Access

each message represents its length. Tasks mapped on different nodes communicate through ST messages and an ST slot should be able to accommodate the longest message transmitted by the associated node. The figure shows how the lengths of the slots associated with N_1 and N_2 are modified after a task has been remapped. In one case, task τ_2 is moved from N_2 to $N_1$and therefore, the message $m_{1,2}$ will disappear (τ_1 and τ_2 are both mapped on N_1), while message $m_{2,4}$ will be transmitted in $Slot_1$ instead of $Slot_2$. In the second case, τ_3 is moved from N_2 to N_1, which means that $m_{1,3}$ disappears, while $m_{3,4}$ is transmitted in $Slot_1$.

4.3.4 BUS ACCESS OPTIMISATION

It may be the case that even after the mapping and partitioning step, some ET activities are still not schedulable. In the third step (lines 10-12, Figure 4.4), our algorithm tries to remedy this problem by changing the parameters of the bus cycle, like ST slot lengths and order, as well as the number, length and order of the ST and DYN phases. The goal is to generate a bus access scheme which is adapted to the particular task configuration. The heuristic is illustrated in Figure 4.8. The algorithm iteratively looks for the right place and size of $Slot_i$ used for transmission of ST messages from $Node_i$ (outermost loops). The position of $Slot_i$ is swapped with all the positions of slots of higher order (line 03). Also, all alternative lengths (lines 04-05) of $Slot_i$ larger than its minimal allowed length (which is equal to the length of the largest ST message generated by a task mapped on $Node_i$) are considered. For any particular length and position of $Slot_i$, alternative lengths of the adjacent ET phase Ph_i are considered (innermost loop). For each alternative, the schedulability analysis evaluates cost $DSch$, and the solution with the lowest cost is selected. If $DSch \leq 0$, the system is schedulable and the heuristic is stopped.

It is important to notice that the possible length π of an ET phase (line 06) includes also the value 0. Therefore, in the final

```
01 for i = 1 to NrNodes
02    for j = i to NrNodes
03        swap Slot_i with Slot_j
04        for all slot lengths λ > min_len(Slot_i)
05            len(Slot_i) = λ
06            for all DYN phase lengths π
07                len(Ph_i) = π
08                if DSch ≤ 0 then stop endif
09                keep solution with lowest DSch
10            end for
11        end for
12        swap back Slot_i and Slot_j
13    end for
14    bind best position and length of Slot_i
15    bind length of Ph_i
16 end for
```

Figure 4.8: Bus Access Optimization

bus cycle, it is not needed that each static slot is followed by a dynamic phase (see also Figure 2.1). Dynamic phases introduced as result of the previous steps can be eliminated by setting the length to $\pi = 0$ (such a transformation is illustrated in Figure 4.3.c). It should be also mentioned that enlarging a slot/phase can increase the schedulability by allowing several ST/DYN messages to be transmitted quickly immediately one after another. At the same time, the following slots are delayed, which means that ST messages transmitted by nodes assigned to upcoming slots will arrive later. Therefore, the optimal schedulability will be obtained for slot and phase lengths which are not tending towards the maximum. The number of alternative slot and phase lengths to be considered by the heuristic in Figure 4.8 is limited by the following two factors:

1. The maximum length of a static slot or dynamic phase is fixed by the technology (e.g. 32 or 64 bits).
2. Only frames consisting of entire messages can be transmit-

ted, which excludes several alternatives.

4.4 Experimental Results

In order to evaluate the proposed heuristic, we have generated a large set of applications with different characteristics. All experiments were run on an AMD Athlon 850 MHz PC. For our first experiments we considered an architecture consisting of 6 nodes. We have generated 4 sets of applications composed of 60, 75, 90, and 120 tasks respectively. Each set consists of 40 applications. The number of unmapped tasks was between 10 and the total number of tasks in the application. Ten task graphs are considered to be unassigned to any of the two domains (ET and TT). The average load on the processors is 60%. The scheduling algorithm MxS3 (Section 3.5.3) has been used for all the experiments presented in the rest of this section

Figure 4.9 shows the percentage of schedulable applications obtained after the successive steps of our heuristic. By straight forward configuration we mean the mapping, partitioning and bus cycle generated at the start of step 1 (line 01 in Figure 4.4). This is a configuration which, in principle, could be elaborated by a careful designer without the aid of optimisation tools like the one proposed in the thesis. Out of the total number of applications consisting of 60 tasks, for example, only 10% were sched-

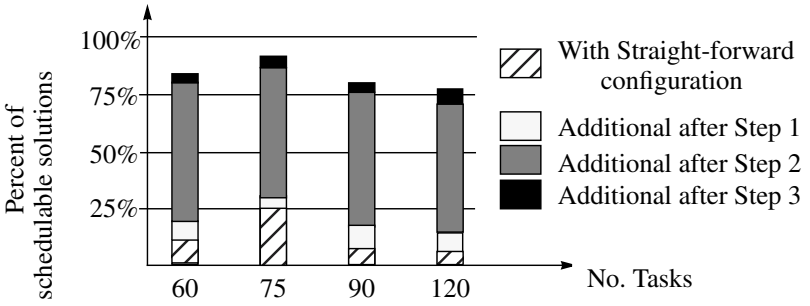


Figure 4.9: Percentage of Schedulable Applications

ulable with the straight-forward configuration and 90% continued the optimisation process. 9% of the total number of tasks have been found schedulable with the configuration generated by step 1. As expected, the mapping, partitioning and bus cycle adjustment performed in step 2 are leading to a huge improvement, adding 61% of the total number of applications to the group of schedulable ones. An additional 4% of the total number of applications is found schedulable after performing the bus optimisation in step 3. A similar trend is followed in the experiments with 75, 90 and 120 tasks. It is easy to observe that by performing the proposed optimisations, huge improvements over the straight-forward configuration has been produced.

An interesting question is to what extent the partitioning of tasks into the ET and TT domains is contributing to the results illustrated in Figure 4.9. Or, are these results mostly due to the optimised mapping? The same question can also be put relative to the bus cycle optimisation. In order to answer these questions, we considered a second set of applications consisting of 60, 80 and 100 tasks grouped in 12, 15, 18, and 20 task graphs and mapped on 4, and 6 nodes. We have run our heuristic for each of these applications considering 4 cases. First, with a subset of tasks that have to be partitioned but no tasks to be mapped ($|\Psi^M| = 0$). Second, with the same subset of tasks open for mapping but not for partitioning ($|\Psi^P| = 0$). The third case does not allow any bus access optimisation, so we switched off the optimisations in lines 5 and 11 in Figure 4.4 (however, we kept the bus cycle adjustment which is needed in Step 2, line 8 in Figure 4.6). The fourth case represents the reference, the complete heuristic. The results are presents in Figure 4.10, which shows the percentage of schedulable applications (relative to the total number of applications) that have been produced by each optimisation step. For example, after step 2, 45% additional applications were schedulable if we only allow to perform re-mapping ($|\Psi^P| = 0$), as opposed to 74% in the case when both optimisations are performed. The same number is 40% if we only allow to perform re-

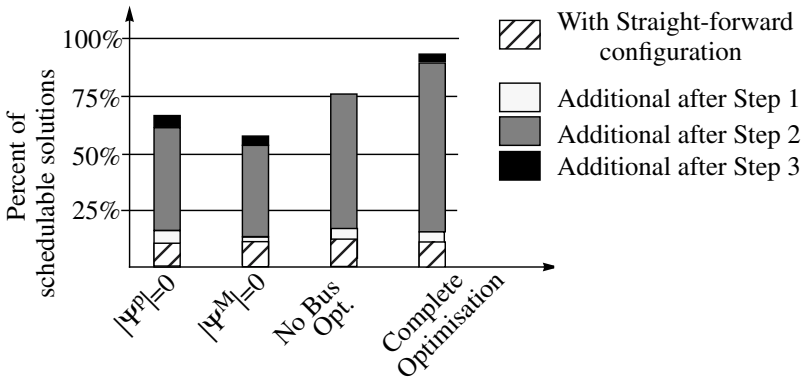


Figure 4.10: Partial Optimisations

partitioning ($|\Psi^M| = 0$). The percentage of unschedulable tasks after the three steps is 34% when $|\Psi^P| = 0$, 44% for $|\Psi^M| = 0$, and 24% when no bus optimisation was performed, as compared to 7% in the case of the complete heuristic. The conclusions which we can draw are the following:

1. An efficient partitioning into the ET and TT domains contributes essentially to the overall optimisation, to an extent comparable to that of an efficient mapping.
2. When applied together, the three techniques provide much better results than the ones obtained when any of the techniques is eliminated.

Concerning the runtime needed for the optimisation process, we have analysed each of the three steps separately. For the examples in Figure 4.9, the average run time for step 1 was 11.7s (60 tasks), 40.1s (75 tasks), 73.2s (90 tasks), and 150s (120 tasks).

The execution time for step 2 is presented in more detail in Figure 4.11 and Figure 4.12. Figure 4.11 illustrates the time needed for step 2 as a function of the total number of task graphs to be partitioned (the characteristics of the applications and the number of nodes are shown in the figure). The upper curve illustrates the average execution times for those applications which

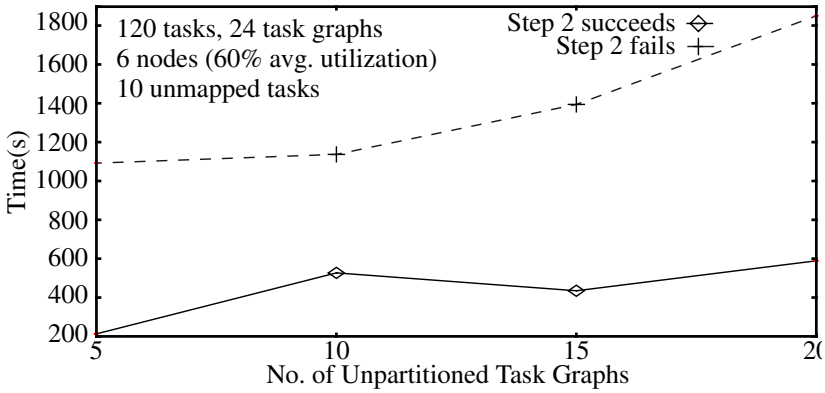


Figure 4.11: Runtime of Step 2 function of $|\Psi^P|$

are running through step 2 without reaching a system configuration which makes them schedulable. This curve can be considered as an upper bound for the execution time in step 2. The second curve in Figure 4.11 gives the average execution times of those applications that have been found schedulable during step 2. In Figure 4.12 we show, in a similar way, the average execution times as a function of the number of unmapped tasks. The execution times needed for the third optimisation step are given in Figure 4.13. As this step is concentrating only on the communication aspect, the average execution time is given as a function of the number of nodes.

Finally, we considered a real-life example from the automotive area, implementing a vehicle cruise controller and a control application related to the Anti Blocking System on an architecture consisting of 5 nodes. The cruise controller consists of 42 tasks organised in 11 task graphs. One of these task graphs is fixed into the TT domain, and the other 10 are unpartitioned. 10 out of the 42 tasks are unmapped. The ABS system consists of 35 tasks already mapped over the 5 nodes and assigned to the ET domain. Running our optimisation heuristic, step 1 was able to generate a correct static schedule for the TT domain, but

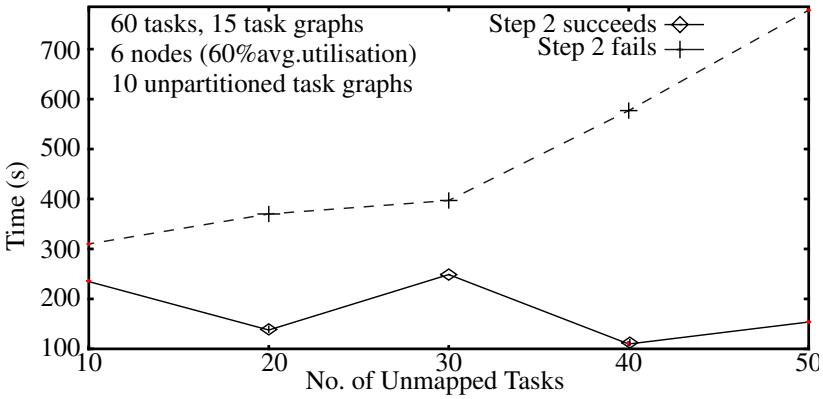


Figure 4.12: Runtime of Step 2 function of $|\Psi^M|$

without producing a globally schedulable system. Step 2 manages to improve the degree of schedulability of the system (function *DSch*) by two orders of magnitude without, however, producing a schedulable system. A correct implementation has been produced after the bus optimisation in step 3. It is interesting to mention that for the final schedulable solution, out of the 10 unpartitioned task graphs, 2 were assigned to the ET and 8 to the TT partition. The run times for the three optimisation steps were 5.3s, 708s and 164s respectively.

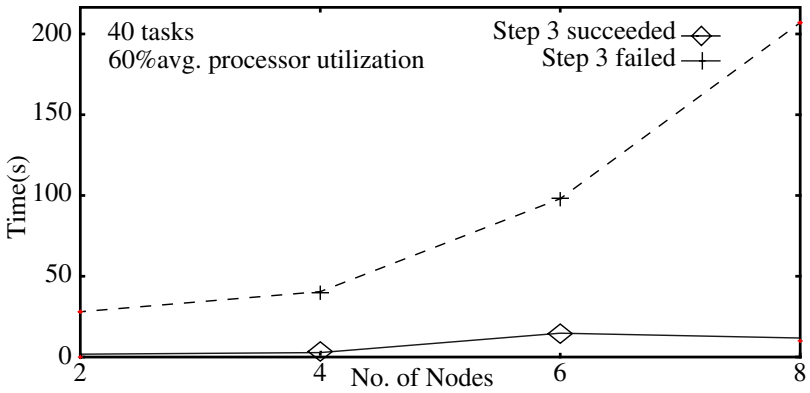


Figure 4.13: Runtime of Step 3

Chapter 5
Conclusions and Future Work

5.1 Conclusions

In this thesis we have first defined and solved the problem of scheduling heterogeneous ET/TT distributed embedded systems. We have proposed several alternative solutions for the scheduling algorithm and we have run extensive experiments in order to compare the efficiency of the alternatives.

Once we have solved the scheduling and schedulability analysis problem, we addressed several design problems which we identified as specific for the type of heterogeneous systems we are studying: the partitioning of functionality into ET and TT activities, and the synthesis of parameters of the mixed ST/DYN communication protocol. These two aspects have been approached in a larger design context, which also involved mapping and scheduling of the functionality. Our experiments have shown the importance of each of the optimisation aspects taken into consideration and the efficiency of the proposed optimisation heuristic.

5.2 Future Work

Several developments are possible in order to generalise the presented approach. First, the communication model can be further generalised such that TT tasks can communicate through DYN messages and ET tasks through ST messages. Then, the interaction mechanism for interaction between TT and ET activities can be refined, in the context of hard-real-time communication.

Based on such a generalised model, we intend to study the problem of priority assignment for ET tasks and DYN messages in the context of mixed TT/ET systems.

The generalised model will also allow us to relax the constraints imposed on the partitioning problem, which in the current version assumes that entire task graphs are moved to the TT or ET domain.

References

[ARI629] ARINC, “Multi-Transmitter Data Bus, Part 1, Technical Description”, ARINC Document 629P1-4, Aeronautical Radio, Inc., Annapolis, MD, USA, 1995.

[Abd99] T. F. Abdelhazer, K. G. Shin, “Combined Task and Message Scheduling in Distributed Real-Time Systems”, IEEE Transactions on Parallel and Distributed Systems, 10(11), pages 1179-1191, 1999.

[Agr94] G. Agrawal, B. Chen, W. Zhao, S. Davari, “Guaranteeing Synchronous Message Deadlines with the Timed Token medium Access Control Protocol”, IEEE Transactions on Computers, 43(3), pages 327-339, 1994.

[Alm99] L. Almeida, “Flexibility and Timeliness in Fieldbus-based Real-Time Systems“, Ph. D. Thesis, University of Aveiro, Portugal, 1999.

[Alm02] L. Almeida, P. Pedreiras, J. A. G. Fonseca, “The FTT-CAN Protocol: Why and How”, IEEE Transactions on Industrial Electronics, 49(6), pages 1189-1201, 2002.

[Aud93] N. Audsley, K. Tindell, A. et. al., “The End of Line for Static Cyclic Scheduling?”, 5th Euromicro Workshop

on Real-Time Systems, 1993.

[Aud95] N. Audsley, A. Burns, et. al., “Fixed Priority Preemptive Scheduling: An Historical Perspective”, Real-Time Systems, 8(2/3), 1995.

[Bal97] F. Balarin, editor, Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Publishers, 1997.

[Bal98] F. Balarin, L. Lavagno, et. al., “Scheduling for Embedded Real-Time Systems”, IEEE Design and Test of Computers, January-March,1998.

[Bec98] J. E. Beck, D. P. Siewiorek, “Automatic Configuration of Embedded Multicomputer Systems”, IEEE Transactions on CAD, 17(2), pages 84-95, 1998.

[Ber03] J. Berwanger, M. Peller, R. Griessbach, “A New High-Performance Data Bus System for Safety Related Applications”, http://www.byteflight.com, 2003.

[Bin01] Enrico Bini, Giorgio Butazzo, Giuseppe Butazzo, “A Hyperbolic Bound for the Rate Monotonic Algorithm”, Proceedings of the 13th Euromicro Conference on Real-Time Systems, pages 59-66, 2001.

[Bli98] T. Blicke, J. Teich, L. Thiele, “System-Level Synthesis using Evolutionary Algorithms”, Design Automation for Embedded Systems, 4(1), pages 23-58, 1998.

[Bos91] R. Bosch GmbH, “CAN Specification Version 2.0”, 1991.

[But97] Giorgio C. Butazzo, “Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications”, Kluwer Academic Publishers, 1997.

[Coff72] E.G. Coffman Jr., R.L. Graham, “Optimal Scheduling for two Processor Systems”, *Acta Informatica*, 1, 1972.

[Dav99] B. P. Dave, G. Lakshminarayana, N. K. Jha, “COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, pages 92-104, 1999.

[Dem01] T. Demmeler, P. Giusto, “A Universal Communication Model for an Automotive System Integration Platform”, Design, Automation and Test in Europe (DATE’01) Conference, Munich, pages 47-54, 2001.

[Dob01a] R. Dobrin, G. Fohler, “Implementing Off-Line Message Scheduling on Controller Area Network (CAN)”, Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, 1, 2001.

[Dob01b] R. Dobrin, G. Fohler, P. Puschner, “Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling”, Proceedings of Real-Time Systems Symposium, 2001.

[Eke00] C. Ekelin, J. Jonsson, “Solving Embedded System Scheduling Problems using Constraint Programming”, Chalmers University of Technology, Sweden, Report number TR 00-12, 2000.

[Ele00a] P. Eles, A. Doboli, P. Pop, Z. Peng, “Scheduling with Bus Access Optimization for Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, 8(5), pages 472-491, 2000.

[Ele02] P. Eles, Lecture Notes for System Design and Methodology, http://www.ida.liu.se/~TDTS30, 2002.

[Erm97] H. Ermedahl, H. Hansson, M. Sjödin, “Response Time Guarantees in ATM Networks”, Proceedings of Real-Time Systems Symposium, 1997.

[Ern97] R. Ernst, W. Ye, “Embedded Program Timing Analysis Based on Path Clustering and Architecture Clas-

sification", Proceedings of International Conference on CAD, pages 598-604, 1997.

[Ern98] R. Ernst, "Codesign of Embedded Systems: Status and Trends", IEEE Design&Test of Comp., April-June, 1998.

[Edw97] S. Edwards, L. Lavagno, E. A. Lee, A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation and Synthesis", Proceedings of the IEEE, 85(3), pages 366-390, 1997.

[Edw00] S. Edwards, "Languages for Digital Embedded Systems", Kluwer Academic Publishers, 2000.

[Fle03] FlexRay homepage: http://www.flexray-group.com/, 2003.

[Fuh00] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, M. Walther, Robert Bosch GmbH, "Time-Triggered Communication on CAN (Time-Triggered CAN- TTCAN)", Proceedings of International CAN Conference, Amsterdam, The Netherlands, 2000.

[Gon91] M. González Harbour, M. H. Klein, J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority, Proceedings of 12th IEEE Real-Time Systems Symposium, pages 116 - 128, 1991.

[Gut95] J. J. Gutiérrez García, M. González Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems", Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, pages 124-132, 1995.

[Han97] H. Hansson, M. Sjödin, K. Tindell, "Guaranteeing Real-Time Traffic Through an ATM Network", Proceedings of the 30th Hawaii International Confer-

ence on System Sciences, 5, 1997.

[IEEE83] EEE Standards Board, “Token Passing Bus Access Method and Physical Layer Considerations“, IEEE Standard 802.4, 1983.7, 1983.

[IEEE98] IEEE Standards Board, “Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications (1998 Edition)“, IEEE Standard 802.3, 1998.

[Jon97] J. Jonsson, K. J. Shin, “A Parametrized Branch-and-Bound Strategy for Scheduling Precedence-Constrained Tasks on a Multiprocessor System“, Proceedings of the International Conference on Parallel Processing (ICPP), pages 158-165, 1997.

[Jor97] P. B. Jorgensen, J. Madsen, “Critical Path Driven Cosynthesis for Heterogeneous Target Architectures”, Proceedings of the 5th International Workshop on Hardware-Software Co-design, pages 15-19, 1997.

[Keu00] K. Keutzer, S. Malik, A. R. Newton, “System-Level Design: Orthogonalization of Concerns and Platform-Based Design”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(12), 2000.

[Koo02] P. Koopman, “Critical Embedded Automotive Networks”, IEEE Micro, 22(4), pages 14-18, 2002.

[Kop92] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, R. Zainlinger, “The Programmer’s View of MARS”, Proceedings of 13th IEEE Real-Time Systems Symposium, pages 223-226, 1992.

[Kop97] H. Kopetz, “Real-Time Systems - Design Principles

for Distributed Embedded Applications", Kluwer Academic Publisher, 1997.

[Kuc97] K. Kuchcinski, "Embedded System Synthesis by Timing Constraint Solving", Proceedings of the International Symposium on System Synthesis, pages 50-57, 1997.

[Lav99] L. Lavagno, A. Sangiovanni-Vincentelli, E. Sentovich, "Models of Computation for Embedded System Design", A. A. Jerraya and J. Mermet eds: System Level Synthesis, Kluwer, 1999.

[Lee99] C. Lee, M. Potkonjak, W. Wolf, "Synthesis of Hard Real-Time Application Specific Systems", Design Automation for Embedded Systems, 4(4), pages 215-241, 1999.

[Leh89] J. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour", Proceedings of 11th Real-Time Systems Symposium, pages 166-171, 1989.

[Leh90] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", Proceedings of 11th IEEE Real-Time Systems Symposium, pages 201 -209, 1990.

[Leu82] J. Y. T. Leung, J. Whitehead, "On the Complexity of Fixed Priority Scheduling of Periodic, Real-Time Tasks", Performance Evaluation, 2(4), pages 237-250, 1989.

[Liu73] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM, 20(1), pages 46-61, 1973.

[LIN00] Local Interconnect Network, "LIN Specification Package", Revision 1.2, http://www.lin-subbus.org/, 2000.

[Lön99] H. Lönn, J. Axelsson, “A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications”, Proceedings of the 11th Euromicro Conference on Real-Time Systems, pages 142-149, 1999.

[Loc92] C. Douglas Locke, “Software Architecture for Hard-Real Time Applications: Cyclic Executives vs. Fixed Priority Executives”, Journal of Real-Time Systems, 4, pages 37-53, 1992.

[Mar00] G. Martin, “The Future of High-Level Modelling and System Level Design: Some Possible Methodology Scenarios”, 9th IEEE/DATC Electronic Design Processes Workshop, 2000.

[Mil99] D. Millinger, P. Gansterer, “The Time-Triggered Operating System Technical Manual”, http://www.vmars.tuwien.ac.at/~fstue/manuals/ttos/doku.html, March, 2003.

[Mic97] G. de Micheli, R. K. Gupta, “Hardware/Software Co-Design”, Proceedings of the IEEE, 85(3), pages 349-365, 1997.

[Pal98] J. C. Palencia, M. González Harbour, “Schedulability Analysis for Tasks with Static and Dynamic Offsets”, Proceedings of the 19th IEEE Real-Time Systems Symposium, pages 26-37, 1998.

[Pal99] J. C. Palencia, M. González Harbour, “Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems”, Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[Ple92] P. Pleinevaux, “An Improved Hard Real-Time Scheduling for IEEE 802.5”, Journal of Real-Time Systems, 4(2), 1992.

[Pop00] P. Pop, P. Eles, Z. Peng, “Bus Access Optimization for

Distributed Embedded Systems based on Schedulability Analysis", Design, Automation and Test in Europe (DATE'00), pages 567-574, 2000.

[Pop01a] P. Pop, P. Eles, T. Pop, Z. Peng, "An Approach to Incremental Design of Distributed Embedded Systems", Proceedings of the 38th Design Automation Conference (DAC), Las Vegas, USA, pages 450-455, 2001.

[Pop01b] P. Pop, P. Eles, T. Pop, Z. Peng, "Minimizing System Modification in an Incremental Design Approach", Proceedings of the 9th International Workshop on Hardware/Software Codesign (CODES 2001), Copenhagen, Denmark, pages 183-188, 2001.

[Pop03a] P. Pop, P. Eles, Z. Peng, "Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems", Real-Time Systems Journal (accepted for publication).

[Pop03b] Paul Pop, Petru Eles, Zebo Peng, "Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems", Design, Automation and Test in Europe (DATE'03) Conference, Munich, Germany, pages 184-189, 2003.

[PopT02] Traian Pop, Petru Eles, Zebo Peng, "Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems", Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES 2002), Estes Park, Colorado, USA, pages 187-192, 2002.

[PopT03a] Traian Pop, Petru Eles, Zebo Peng, "Schedulability Analysis for Distributed Heterogeneous Time/Event-Triggered Real-Time Systems", to be published in the Proceedings of the 15th Euromicro Conference on

Real-Time Systems (ECRTS 2003), Porto, Portugal, July 2-4, 2003.

[PopT03b] Traian Pop. Petru Eles, Zebo Peng, "Design Optimization of Mixed Time/Event Triggered Distributed Embedded Systems", submitted for publication.

[Pra92] S. Prakash, A. Parker, "SOS: Synthesis of Application Specific Heterogeneous Multiprocessor Systems", Journal of Parallel and Distributed Computers, 16, pages 338-351, 1992.

[Raj93] P. Raja, G. Noubir, "Static and Dynamic Polling Mechanisms for Fieldbus Networks", ACM Operating Systems Review, 27(3), 1993.

[Ric02] K. Richter, R. Ernst, "Event Model Interfaces for Heterogeneous System Analysis", Proceedings of Design, Automation and Test in Europe Conference (DATE'02), Paris, France, 2002.

[Ric03] K. Richter, M. Jersak, R. Ernst, "A Formal Approach to MpSoC Performance Verification", IEEE Computer, 36(4), pages 60-67, 2003.

[Sch94] M. Schwehm, T. Walter, "Mapping and Scheduling by Genetic Algorithms", Conference on Algorithms and Hardware for Parallel Processing, pages 832-841, 1994.

[Sha90] L. Sha, R. Rajkumar, J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real Time Synchronization", IEEE Transactions on Computers, 39(9), pages 1175 -1185, 1990.

[Sta94] J. A. Stankovic, M. Spuri, M. di Natale, and G. C. Butazzo, "Implications of Classical Scheduling Results for Real-Time Systems", Technical Report UM-CS-94-089, Computer Science Department, University of Massachusetts, 1994.

[Str89] J. K. Strosneider, T. E. Marchok, “Responsive, deterministic IEEE 802.5 Token Ring Scheduling”, Journal of Real-Time Systems, 1(2), 1989.

[Tab00] B. Tabbara, A. Tabbara, A. Sangiovanni-Vincentelli, “Function/Architecture Optimization and Co-Design of Embedded Systems”, Kluwer Academic Publishers, 2000.

[Tin92] K. Tindell, A. Burns, A.J. Wellings, “Allocating Hard Real-Time Tasks (An NP-Hard Problem Made Easy)”, Journal of Real-Time Systems, 4(2), pages 145-165, 1992.

[Tin94a] K. Tindell, J. Clark, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”, Microprocessing & Microprogramming, Vol. 50, No. 2-3, 1994.

[Tin94b] K. Tindell, H. Hansson, A. J. Wellings, “Analysing Real-Time Communications: Controller Area Network (CAN)”, Proceedings of Real-Time Systems Symposium, 1994.

[Tin94c] K. Tindell, “Adding Time-Offsets to Schedulability Analysis”, Department of Computer Science, University of York, Report Number YCS-94-221, 1994.

[TTP01A] TTP/A Specification, WebSite of Time-Triggered Technology, http://www.tttech.com/, 2001.

[TTP01C] TTP/C Specification, WebSite of Time-Triggered Technology, http://www.tttech.com/, 2001.

[Tur99] J. Turley, “Embedded Processors by the Numbers”, Embedded Systems Programming, 1999.

[Ull75] D. Ullman, “NP-Complete Scheduling Problems”, Journal of Computer Systems Science, 10(3), pages 384-393, 1975.

[Wor03] World FIP fieldbus, http://www.worldfip.org/, April 2003.

[Wol94] W. Wolf, “Hardware-Software Co-Design of Embedded Systems”, *Proceedings of the IEEE*, V82, N7, 1994.

[Wol97] W. Wolf, “An Architectural Co-Synthesis Algorithm for Distributed Embedded Computing Systems”, IEEE Transactions on VLSI Systems, pages 218-229, 1997.

[Wol03] W. Wolf, “A Decade of Hardware/Software Codesign”, IEEE Computer, 36(4), pages 38-43, 2003.

[Xu93] J. Xu, D.L. Parnas, “On satisfying timing constraints in hard-real-time systems”, IEEE Transactions on Software Engineering, 19(1), 1993.

[Yen97] T. Y. Yen, W. Wolf, “Hardware-Software Co-Synthesis of Distributed Embedded Systems”, Kluwer Academic Publishers, 1997.

APPENDIX A
List of Notations

Notation	Description
Γ_i	Task-graph i
τ_{ij}	Activity j belonging to task-graph Γ_i
T_{ij}	Period of activity τ_{ij}
C_{ij}	Worst-case execution time of activity τ_{ij}
D_{ij}	Deadline for activity τ_{ij}
Φ_{ij}	Offset for activity τ_{ij}
J_{ij}	Jitter for activity τ_{ij}
R^b_{ij}	Best-case response time for activity τ_{ij}
R_{ij}	Worst-case response time for activity τ_{ij}
$Prio_{ij}$	Priority for activity τ_{ij}
$Node_{ij}$	The node on which task τ_{ij} is mapped
Tss	LCM of the periods of all the task-graphs in the application
$H_{ij}(w_{ij})$	ET demand associated with the busy period of length w_{ij} of activity τ_{ij}
$A_{ij}(w_{ij})$	ET availability associated with the busy period of length w_{ij} of activity τ_{ij}

Notation	Description
Ψ^{P}	The set of task-graps which can be repartitioned between time-triggered and event-triggered domains
Ψ^{M}	The set of tasks which can be remapped