# Scheduling and Timing Analysis for Safety Critical Real-Time Systems

Submitted for the degree of Doctor of Philosophy

*Iain John Bate*

Department of Computer Science,
University of York.

November 1998

# Abstract

The scheduling of tasks in safety critical systems such that the timing requirements are reliably met is often difficult. Safety critical systems are different to other systems because a failure to meet a requirement may lead to a catastrophic effect, for example an accident leading to loss of life. For this reason, there is a greater emphasis on verification and validation of software than with other types of systems. Safety critical systems are found in a wide range of industries including; nuclear, chemical, aerospace and automotive industries. The particular area of interest for this thesis is in the avionics industry and in particular engine controllers. This provides a greater emphasis on hybrid systems, where the software is responsible for the performance of the mechanical components.

Frequently, the systems developed have problems caused by insufficient resources, this leads to costly redesigns and timing requirements having to be altered so that they can be met. A major contributor to the problems is the nature of the cyclic scheduler that is often used. When this work began, fixed priority scheduling was proposed as a potential solution to the problems. However, additional work was required for fixed priority scheduling so it could be used in practice. The areas of work covered are: task attribute assignment, the development of a new or modification of an existing infrastructure, and the production of appropriate timing analysis. Through the course of the thesis, solutions are developed for the required areas of work for both uniprocessor and distributed systems.

As a measure of success Rolls-Royce have used the work on an actual engine controller. This followed the Joint Aviation Authority (JAA), which is the relevant certification authority, approving its use. In addition, the Guards project (which is a multi-company ESPRIT funded collaboration developing new techniques for critical real-time systems) is also using parts of the analysis developed. The fact that technology transfer has been achieved is an indication of its acceptability to industry.

# Contents

# List of Figures

# List of Tables

# Table of Abbreviations

| **Abbreviation** | **Meaning** |
|---|---|
| SPARK | SPade Ada Real-time Kernel |
| IMA | Integrated Modular Avionic |
| AOCS | Attitude and Orbital Control System |
| TTA | Time Triggered Architecture |
| TDMA | Time Division Multiple Access |
| FIFO | First-In First-Out |
| COTS | Commercial Off-The Shelf |
| DMPO | Deadline Monotonic Priority Order |
| LCM | Least Common Multiple |
| JAA | Joint Air-Worthiness Authority |
| COMP | Composite task |
| WCRT | Worst-Case Response Task |

# Table of Symbols

| Symbol | Meaning |
|---|---|
| $C_i$ | worst-case execution time of task $i$ |
| $T_i$ | period of task $i$ |
| $U_{max}$ | maximum processor utilisation |
| $R_i$ | worst-case response time of task $i$ |
| $R_i^n$ | $n^{th}$ iteration of the calculation of $R_i$ |
| $B_i$ | blocking time for task $i$ |
| $I_i$ | interference time for task $i$ |
| $D_i$ | deadline of task $i$ |
| $w_i$ | worst-case response time of task $i$ |
| $w_i^n$ | $n^{th}$ iteration of the calculation of $w_i$ |
| $p_i$ | priority of task $i$ |
| $J_i$ | release jitter for task $i$ |
| $n_i$ | instance being analysed of task $i$ |
| $clk$ | task representing the overheads due to the clock tick occurring |
| $first$ | task representing the overhead of releasing the first task at a particular time |
| $sub$ | task representing the overhead of releasing tasks subsequent to task $first$ at a particular time |
| $S_i$ | separation requirement for task $i$ |
| $O_i$ | offset for task $i$ |
| $T_D$ | end-to-end deadline requirement of a transaction |
| $R_J$ | worst-case response time calculated using the release jitter approach |
| $R_E$ | worst-case response time calculated using the exact approach |
| $R_C$ | worst-case response time calculated using the composite offset approach |

# Acknowledgements

I would like to thank my colleagues in the Real-Time Systems and High Integrity Systems Engineering groups at the University of York, and the engineers at Rolls-Royce for the fruitful discussions that have taken place. In particular, I acknowledge the support of Tim Kelly, Professor Andy Wellings, Dr Andrew Vickers (now with Praxis Critical Systems), Dr Ben Whittle (now with Union Bank of Switzerland), and Stuart Hutchesson of Rolls-Royce. The main thanks goes to Professor Alan Burns (my supervisor), Professor John McDermid, and Kate Utting, without their support and tolerance I could not have completed the work for this thesis. The work was performed under contract for Rolls-Royce plc, whose support I am grateful for.

# Declaration

Certain parts of this thesis have appeared in previously published papers specifically the following references (marked * for principal author):

Technology Transfer: An Integrated 'Culture Friendly Approach' Approach, I.J. Bate* et al, Proceedings of Technology Transfer Workshop - Part of the 18th International Conference on Software Engineering, Berlin, Germany, 29-30th March 1996.

Putting Fixed Priority Scheduling Theory into Engineering Practice for Safety Critical Systems, I.J. Bate*, A. Burns, N.C. Audsley, Proceedings of the 2nd Real-Time Applications Symposium, pp. 2-10, Boston. USA, June 1996.

Towards a Fixed Priority Scheduler for an Aircraft Application, I.J. Bate*, A. Burns, J.A. McDermid, A.J. Vickers, Proceedings of the 8th Euromicro Conference on Real-TIme Systems, IEEE System Press, pp. 34-40, L'Aquilla, Italy, June 1996.

Flexible Scheduling for Advanced Engine Controllers, I.J. Bate*, A. Burns, N.C. Audsley, IEE Colloquia on Hybrid Control in Real-Time Systems, 12th December 1996

Flexible Scheduling for Engine Controllers, I.J. Bate*, A. Burns, The Patent Office, UK Patent Application Number 9710522.5 and US Patent Application Number 5,606,695, May 1997.

Schedulability Analysis of Fixed Priority Real-Time Systems with Offsets, I.J. Bate*, A. Burns, Proceedings of the 9th Euromicro Workshop on Real-Time Systems, IEEE System Press, Toledo, Spain, June 1997.

Building a Preliminary Safety Case: An Example from Aerospace, I.J. Bate*, A. Burns, T.P. Kelly*, J.A. McDermid, Proceedings of the 1997 Australian Workshop on Industrial Experience with Safety Critical Systems and Software, October 1997.

A Dependable Distributed Architecture for a Safety Critical Hard Real-Time System, I.J.Bate*, A. Burns, IEE Colloquium on Hardware Systems for Dependable Applications, November 1997.

The Role of Timing Analysis in the Certification of IMA Systems, N. Audsley, I. Bate, A. Grigg, IEE Symposium on Certification of Ground/Air Systems, February 1998.

Investigation of the Pessimism in Distributed Systems Timing Analysis, I.J. Bate*, A. Burns, 10th Euromicro Workshop on Real-Time Systems, June 1998.

# Chapter 1

# Introduction

Safety-critical systems are different to other systems because a failure to meet a requirement may lead to a catastrophic effect, e.g. an accident leading to loss of life. An example definition of "catastrophic" is taken from MIL STD 882C [1] "Death, system loss or severe environmental damage". For this reason, there is a greater emphasis on verification and validation of software in safety-critical systems than with other types of systems.

The importance of verification is indicated by the fact over 50% of the software effort for the Boeing 777 has been in the areas of analysis and testing [2, 3]. Typically, the costs of producing safety critical systems are significantly higher than for other domains. Therefore, there is a greater potential for process improvement initiatives because the savings have the potential to pay for the investment costs. Safety critical systems are found in a wide range of industries, including; nuclear, chemical, aerospace and automotive industries. The particular domain of interest for this thesis is avionics and in particular engine controllers.

The aspects of safety critical systems being considered in this thesis are those that either control or supervise the operation of the system's functionality. Until relatively recently (in the context of safety critical systems, recent could be considered to be 10-20 years) control systems were implemented entirely in hardware using hydro-mechanical operations. There are a number of problems with hardware solutions, including: the cost of producing systems is high, the hardware that provides the control is large and heavy, and the hardware has a finite slew rate performance that limits the system's responsiveness [4].

The need for cheaper operating costs during the lifetime of the system has lead to more advanced controllers with reduced weight being used. This need has seen the gradual introduction of computers to form embedded systems that provide hybrid control. Hybrid control is where an embedded system is used to make the calculations used in the control of the system. Computer controlled systems have the advantage that a great deal of functionality can be implemented on a single processor, leading to reduced cost and weight. An added benefit is that software is a non-recurring cost due to the fact it is only produced once (ignoring the effects of maintenance), whereas hardware adds cost to every system produced.

In comparison to many established engineering disciplines, such as control systems engineering, software engineering can be considered to be in its infancy, particularly for safety critical systems. Part of the reason for the high cost of safety critical systems is that the development of these systems is prone to a large amount of change. The systems are susceptible to change because the systems are produced by a concurrent life-cycle, where parts of the system are produced before others dependent parts are fully considered. Production is concurrent because some parts of the system have longer supply lead times than others. As the different parts of the system evolve there is likely to be change to the other parts of the system.

One of the activities in the implementation and verification of safety-critical systems is the meeting of the timing requirements. The mechanism for controlling the system's timing behaviour is through a scheduler, which dictates the run-time order that functionality units are performed in. This thesis presents research performed on scheduling and timing analysis, in the context of safety critical systems. The term "timing analysis" is taken to mean the overall process of guaranteeing the system's timing requirements are met.

## 1.1 Current Scheduling Practice

An evaluation of industrial practice for avionic systems confirms that the majority of safety critical systems currently employ a cyclic scheduler. The operation of the cyclic scheduler involves two constituent parts; a minor cycle and a major cycle. The major cycle is a sequence of tasks that are executed periodically.

Each major cycle consists of a number of minor cycles that split the major cycle into uniform parts.

The schedule is produced by allocating each task to specific positions in a number of minor cycles so that the timing requirements are met. There are two principal reasons for splitting up the major cycle into minor cycles rather than having a single cycle of tasks. The reasons being the ease of synthesis and the reduction of jitter[1]. The synthesis problem is simplified by a "divide and conquer" strategy in which the system being developed is split into a number of parts. The parts are initially handled separately and later as an integrated unit. Splitting the major cycle into minor cycles reduces jitter because more synchronisation points are provided.

Many cyclic schedulers have similar characteristics, which are largely dictated by the nature of the systems being developed. The characteristics are:

1. *Many tasks* - There is a tendency to have a large number of small tasks (small in terms of their worst-case execution times), more than 50 per processor. The reason is a smaller task is more likely to fit in the available gaps when balancing and packing minor cycles that are almost full. Therefore, the production of a schedule for the system is seen as more manageable. However, the number of tasks is often not representative of the natural decomposition of the software, which leads to maintenance and development problems.

2. *Heavily loaded* - Safety critical systems have a tendency towards high processor utilisation, often almost 100%. This is in part due to the designers needing to try to maximise the use of the available hardware. For a number of reasons, the choice of processor tends to be of an older generation rather than the most powerful up-to-date available. Older generation processors have more operational experience, which increases the likelihood of design errors being known. Therefore, the design of the software can account for discrepancies in the design of the hardware that may otherwise cause problems.

[1]Jitter is the change in the time when a task is released. Jitter is caused by variations in the time tasks take to execute. The jitter increases through the course of the cycle, starting with a value of zero at the start of the cycle.

For example, the 68020 processor is often chosen instead of the Power PC range, which is technically more advanced and provides greater performance. The reasons are partly because it has been used before in safety critical systems, but also because design errors are already known through its use in many applications. These applications include Personal Computers, photocopiers etc.. Other reasons include; the design is simpler than the Power PC processor making it easier to analyse, the likelihood of MIL-SPEC components being available is increased, and they are cheaper to purchase. The cheaper cost of the processor may seem insignificant when compared to the overall system. However, while a commercial version of the 68020 may only cost a few pounds, a military version costs hundreds of pounds. This is particularly important because hardware is a recurring cost through the lifetime of the system, whereas software is an up-front cost paid for once (except for maintenance). The simpler processor design also helps to ease the certification process.

All the factors point towards a choice of the older generation processor. However, the older generations of processor have less processing capability than more modern processors. In practice, whichever processor is chosen the system is always likely to be heavily loaded since engineers invariably find extra functionality that can be included. Therefore, a more powerful processor may not necessarily solve the problem of the system being heavily loaded.

3. *Single Processor* - Currently, the computing infrastructures used tend not to be distributed throughout the physical system being controlled. Instead, a single processor, which may be replicated, performs the necessary functionality. The benefit of this approach is that a uniprocessor is simpler to certify than a distributed system because it is easier to reason about tasks' interactions when they are located on the same processor. In this context, a distributed system is considered to be a number of processors that communicate using messages via a fully interconnected data bus. The drawback of the single processor approach is that the functionality is not executed where it is needed. For example, a data calculation is performed remotely from the actuator that uses it. Therefore, more cabling is required leading to an increase in weight [5].

Through discussions with engineers from industry, a number of problems with cyclic scheduling have been highlighted. Locke [6] also discusses similar problems with cyclic scheduling. In brief, the following points are highlighted:

1. the cyclic scheduler is hard to maintain due to the difficulty in synthesising the scheduler and poor robustness to change,
2. a great deal of resource is wasted due to the restricted computational model (i.e. only having periodic tasks whose rate is an integer multiple of the minor cycle), and
3. it is difficult to assess whether the system is likely to be schedulable until the final system is available.

These problems are discussed in greater detail in section 2.5.

## 1.2 Alternative Approaches to Scheduling

There are a number of trends that are going to greatly influence the development of future systems, primarily the need for cheaper development and maintenance. A number of papers, for example [7, 8, 9], describe how novel architectural approaches simplify the design and maintenance of systems. This is achieved by facilitating technology transparency (the ability to abstract the system's behaviour from the specific environment), interchangeability (the ability to run the software on a number of platforms), and systematic reuse (to have a process and architecture that encourages the reuse of the system's components). One of the principal enabling technologies for the trends to be successful is the provision of a more flexible and predictable scheduling technique.

Having recognised the need to use a different scheduling approach than the cyclic scheduler, there are two issues to be dealt with. These issues are whether there is a more suitable scheduling approach, and how it should be used within the current framework of safety critical systems.

The largest body of research work into scheduling has been performed on priority-based scheduling, which has two variants; fixed priority and dynamic priority. Both approaches rely on all the tasks having a priority, and that runnable tasks

are executed in priority order. The fundamental difference between the two priority-based approaches is that in fixed priority scheduling each task has a static priority assigned off-line, whereas in dynamic priority scheduling tasks have changing priorities at run-time dependent on a particular metric. For example, in fixed priority scheduling the task priorities could be assigned so the higher priority tasks have smaller periods, which is referred to as rate monotonic scheduling [10]. In dynamic scheduling, the task priorities could be assigned so the higher priority tasks have the closest deadlines, which is referred to as earliest deadline first scheduling [11].

For a scheduling approach to be acceptable in the safety critical domain the technique has to be predictable. In the context of scheduling, predictability is the ability to state at any time the task that will be executed next. A predictable scheduling approach allows analysis that demonstrates the timing requirements are met. This forms part of the justification of the integrity of the system. Predictability is determined by two entities; the method for selecting task priorities and the mechanism for releasing tasks based on the priorities. In these respects both the fixed and dynamic priority based schedulers are predictable. A key point is the worst-case schedule, from a timing perspective, for both priority-based approaches is deterministic. In the context of scheduling, deterministic is the ability to state before execution commences the run-time ordering of tasks.

The principal difference between the cyclic scheduling approach and the priority-based approaches is that the cyclic approach is deterministic (i.e. a static run-time ordering of tasks) and the priority-based approach is only predictable (i.e. there is a dynamic run-time ordering of tasks in general). The safety critical systems industry is very conservative, and fixed priority scheduling is often viewed as difficult to certify [12]. The problem of certifying a fixed priority system is simpler than one using dynamic priority scheduling. Other prohibitors to the use of dynamic scheduling is that the scheduler becomes more complex and overheads larger when compared to techniques where decisions are taken off-line. In contrast, the fixed priority scheduler run-time model is relatively simple.

For the reasons given above, fixed priority scheduling is investigated within this thesis as the solution to the problems of cyclic scheduling. This solution is viewed as a smaller evolutionary step than dynamic priority scheduling. Later in the thesis it will be seen that a simplified version of fixed priority scheduling is

advocated to ease technology transfer problems. In Locke's paper [6], problems of cyclic scheduling are highlighted and fixed priority scheduling is viewed as the "best" (in this context best is the minimal technological change capable of solving the problems) solution to the problems. A key observation about Locke's paper is that it deals with a purely academic scheduling model and does not consider the practicalities of using the technique in industry.

## 1.3 Domain Specific Oppositions to Change

A great deal of academic work is performed that is claimed to be suitable for safety critical systems. However, little of the work ever seems to be adopted by industry [13, 14, 15]. The problem with the work seems to be that it is too general in nature, trying to cope with a number of domains and a wide range of systems [14]. The solutions that result can not possibly support all the necessary constraints of a particular system efficiently or effectively. For this reason, the work in this thesis is specifically targeted at industrial projects developing avionic systems. Taking this strategy allows the development of a much tighter solution, which is fit for the specific system's purpose. However, much of the work has applicability to systems in general.

Avionic systems were chosen over a multitude of other options because it is an application with more information available due to personal work experience with and for British Aerospace plc and Rolls-Royce plc. Having more information available increases the actual relevance of the work. This is particularly important because in-depth details of actual safety critical systems are not generally available in the public domain. Avionic systems are classed as critical, often the highest integrity level, systems.

Avionic systems feature hard real-time requirements. Hard real-time requirements are defined as those that have to be guaranteed to be met. The timing aspects of the system are controlled by a scheduler. Since the scheduler is responsible for controlling when any functionality is performed, it is fundamental to the flexibility and predictability of the system. Also, the scheduler is affected by the maintenance of the system as a whole. As the techniques for producing systems evolve, scheduling must become more advanced. The work presented in

this thesis investigates the evolution of scheduling to support the development of safety critical systems.

Specific oppositions to any change caused by the domain may be considered more important than the resulting technical and commercial benefits. This work recognises these oppositions and adapts the existing body of work on fixed priority scheduling accordingly. Certainly technical benefits alone are not enough to ensure the adoption of any proposed change. In practice, commercial factors (such as cost and risk) normally have a greater influence. Four basic oppositions to any change to how a system is developed are considered:

1. *Certification* - The safety critical domain is traditionally very conservative. Any significant change requires a great deal of work to assess the risks and to show the system is at least as safe as it was before the change. The measure of success is whether sufficient evidence of the approach's integrity can be generated to satisfy the certification authorities.

2. *Sufficiency* - For reasons of cost, any technique that is developed should be flexible and efficient enough to allow the engineer to design, implement and maintain the system with a minimum of effort. The measure of success is whether the set of requirements can be met if it is feasible to do so.

3. *Understanding* - A key, but difficult to assess, criterion is the approach must be understandable to those who must apply it in the industrial context. For the purpose of the thesis, the measure of success for this criterion is whether industry has accepted and/or adopted the work. In addition, it is important that the engineers who use the technique do not need specialist knowledge or significant amounts of re-training.

4. *Reuse* - A great deal of investment exists in the current systems that are in service and also those currently being developed. Wherever possible the change should not affect the rest of the system. Also, the approach should allow the majority of the components of existing systems to be reused. The measure of success is how much of the hardware and software of the system needs to be changed.

In industry, the risks and probable cost associated with a change must be assessed before any decision is taken as to whether the change is acceptable. Therefore,

the changes to current practice within this thesis are considered with respect to the four criteria above.

## 1.4 Thesis Proposition

The central proposition of this thesis is:

*The proposed simplified version of fixed priority scheduling will ease the problem of meeting timing requirements now, and for the immediate future for industrial safety critical embedded systems.*

There are two strands to this thesis. Firstly, the thesis is concerned with supporting the transition from cyclic scheduling to fixed priority scheduling. Secondly, the thesis addresses the transition from uniprocessor to distributed systems.

## 1.5 Thesis Structure

Chapter 2 contains a description of the current role and approach taken for scheduling and timing analysis within the safety critical systems domain. The description is not intended to be an all encompassing statement, instead it is meant as a general purpose description for systems within the domain. The aim of the chapter is to provide a basis for the changes proposed and to explain the problems currently encountered when developing systems.

Chapter 3 contains a literature survey supporting the rest of the document giving background information and the history of fixed priority scheduling.

Chapter 4 considers the transition from cyclic scheduling to fixed priority scheduling. The work addresses the benefits of using fixed priority scheduling and whether the current literature on the subject is sufficient to implement systems of the kind discussed in Chapter 2. Part of Chapter 4 is to derive an appropriate computational model with timing analysis, and investigate how pessimism in the timing analysis may be reduced. (In the context of scheduling, pessimism is defined as where the analysis indicates the system is unschedulable when it is in fact schedulable.) During the course of Chapter 4, two areas of work are identified that need attention to allow complex control systems to be implemented in

the safety critical domain. Firstly, a scheduling approach is required that allows the reuse of the current infrastructure and application, this is derived in Chapter 5. Secondly, a mechanism is needed for assigning attributes to tasks so that the timing requirements are met, this is derived in Chapter 6.

Chapter 7 presents a case study of how the techniques have been applied to a real system in order to demonstrate that the concepts are appropriate and capable. The system chosen currently uses a cyclic scheduler.

Chapter 8 presents timing analysis for task sets where tasks feature offsets. An offset is a delay from a known reference of when a task is released. The approach provides a solution to the need for timing analysis, which features low pessimism, low computational complexity and is understandable.

Chapter 9 addresses the transition from uniprocessor-based systems to a distributed system. This chapter provides a computational model for scheduling within the distributed system, and then proceeds to develop timing analysis with relatively low computational complexity and pessimism. Extensive simulation is used to evaluate the derived solution. The timing analysis approach is based on the uniprocessor timing analysis in Chapter 4. However, the task attribute assignment technique has been adapted to improve schedulability and robustness to change. The work uses the offset analysis developed in Chapter 8.

Finally, Chapter 10 summarises the achievements contained in this thesis and provides suggestions for further avenues of research.

# Chapter 2

# Current Approaches to Scheduling in Safety Critical Systems

The purpose of this chapter is to define the current techniques and mechanisms employed for scheduling and timing analysis within the safety critical systems domain. The aim is to provide a baseline so that the effect of the transitions can be properly assessed later. The transitions are from uniprocessor cyclic scheduling to fixed priority scheduling, and then to scheduling for distributed systems. The work is aimed at general safety critical systems, with particular emphasis being given to avionic systems.

The chapter introduces the requirements relevant to scheduling within safety critical systems. Typically the scheduling requirements are of two types: application specific (in this case avionic systems that provide control or supervision of operation), and domain specific (in this case, general constraints for safety critical systems). Related to producing a scheduler, there are three principal categories of requirements: timing, functional, and safety. It is assumed that appropriate techniques already exist for guaranteeing the functional correctness of the scheduler. The work covers timing and aspects of safety where it is related to timing. There are four main parts to the discussion, which are; timing requirements (section 2.1), certification requirements (section 2.2), process issues (section 2.3) and infrastructure details (section 2.4).

The final part of the chapter, section 2.5, contains a detailed discussion of the specific problems of using cyclic schedulers.

## 2.1 Timing Requirements to be Met

Most of the safety critical embedded systems are developed for commercial applications, and hence their details are not publicly disclosed. Therefore, realistic examples of timing requirements are hard to find. The purpose of this section is to discuss the general characteristics of the system to be implemented, and its domain, to establish the typical timing requirements.

The discussion uses the contents of three main sources, which are: the DICOSMOS project by Torngren [16], the General Avionic System [17], and the Magneto Stereotaxis System by Wika and Knight [18]. The three sources are used as examples to derive the typical timing characteristics, which are then summarised in section 2.1.4. The discussion is supplemented with information obtained through a number of years experience working for the avionics industry. Considering these three sources, it is possible to provide an outline of the typical timing requirements. During the course of the discussion it is necessary to introduce a few control system concepts in order to place the requirements in context.

### 2.1.1 DICOSMOS

The paper by Torngren [16] discusses the general timing requirements found in the DICOSMOS project. The aim of the project is to investigate theories and design rules needed when applying distributed computer-based solutions. The systems are used to control complex machinery, such as industrial robots, production machines, vehicles, aircraft, etc.. Torngren's work does not specifically consider the needs of safety critical systems. However, many of the systems discussed would invariably be classed as critical, i.e. high integrity. Therefore, the requirements of this project are relevant.

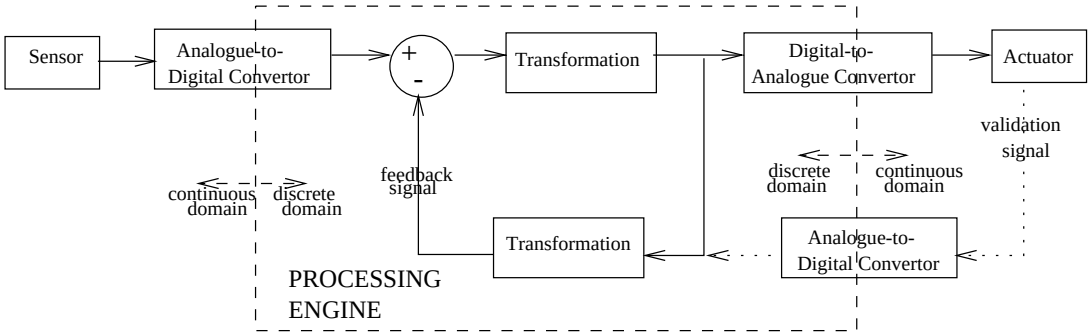


Figure 2.1: A Typical Control Loop for An Embedded System

### Overview of Basic Control Systems

Torngren defines a control system using three basic modelling entities: transformation, sampling, and feedback. The type of system is illustrated in Figure 2.1. Figure 2.1 illustrates two transformations, which are the principal means for controlling the operation of the system. The transformation effectively applies a mathematical function to any signal that is input. The transformations are part of a feedback loop.

The intention of having a feedback loop is to allow any new output to be based on the current output as well as the latest input. Two routes for the feedback signal are illustrated; one is to use the signal that is output to the actuator and the other is to read the actual actuator value. The benefit of using the actual actuator value is that the system's characteristics (such as inertia) that cause non-ideal operation are accounted for. The feedback path also allows error checking of the actuator's operation. However, the drawback is that extra requirements and functionality are needed.

Feedback is used so that the rate of change of the output can effectively be damped (i.e. slowed down), which increases the stability of the output signal and reduces the impact of any infrequent signal errors, e.g. spikes on the signal. Stability is a measure of the ease in which a signal can be affected leading to unpredictable behaviour. The stability of any control system is considered important. Stability should be controlled so that the system performs as expected. Poor stability control could result in the system going out of its operational bounds, leading to mechanical damage. However, too much control over stability reduces the performance of the system by constraining the response to

stimuli. The optimum performance is obtained through operating the system at the limit of its stability [19].

The sampling stages in Figure 2.1, analogue-to-digital and digital-to-analogue conversion, allow the system to convert between the continuous and discrete domains. Continuous is defined as *connected throughout in space or time* [20], i.e. the value always has the ability to change. Continuous systems process analogue signals. An analogue signal is one that can assume any value within a permitted range. Before the use of computers, control systems were entirely produced with continuous signals using hydro-mechanical components. Discrete is described as *discontinuous* [20], i.e. where there is a finite interval during which a value does not change. Discrete components, such as microprocessors, are used to process digital signals. A digital signal has a finite set of values that can be chosen. The simplest form is a binary signal where there are two values zero or one that relate to true or false.

Figure 2.1 shows the transformations are performed based on discrete values. The tasks that complete the path from the sensor input through the transformation function to actuator output within the processing engine would be represented as a transaction. A transaction is where a number of tasks are executed in a pre-defined order, normally within a specified time limit. The transaction would contain a sequence (data capture, followed by transformation, followed by data output) and an end-to-end deadline (to constrain latency). Outside of the transformation stages and feedback loops is the actual system, which relies on information being in a continuous form.

### Problems with the Discrete Domain

Torngren makes a number of observations concerning the relationship between discrete time control systems (such as programmable electronic control systems) and continuous time control systems (such as traditional analogue systems). For a discrete control system, a classical result is that time delays deteriorate performance from the ideal [21], for example noise is introduced onto the signal. A drawback of moving to computer controlled system is that calculations and actions no longer occur virtually instantaneously. Instead, signals have both latency and jitter. Latency is the delay from the first point in time when a value

could have arrived until when the signal actually does arrive. For an output signal, the latency is in effect from the time just after the generation of the initial input signal until the actual output. The jitter defines the maximum variation in periodicity of a function. Jitter can also be considered as the variation in latency. The purpose of defining latency and jitter is to provide a constraint over any instability. There is a direct relationship between stability and jitter - as jitter increases stability is reduced.

An example of the effect of jitter is shown using the differentiation function given in the left hand side of equation (2.1). The differentiation function is frequently used in control loops and is defined as the rate of change of a function $f(x)$ with respect to time. The value of the differentiation function is normally dependent on the value $x$. Typically the function is implemented in the discrete domain based on regularly sampled values using the approximation given in the right hand side of equation (2.1).

$$\frac{\delta f(x)}{\delta t} \approx \frac{\Delta f(x)}{\Delta t} \tag{2.1}$$

where $\Delta f(x)$ is the change in value between samples, and
$\Delta t$ is the time between samples.

In a cyclic schedule, the sampling function is implemented with a periodic task. The difference $\Delta f(x)$ is calculated from successive samples and the sampling rate is assumed to be a constant. However, the effect of jitter is to cause the value of $\Delta t$ to change. For control algorithms, it is considered difficult to allow for varying $\Delta t$ (i.e. changing the time between samples, this is referred to as multi-rate control) [22]. The difficulty is that the trim values (i.e. gains of amplifiers) for the transformation function must account for changing rates. Therefore, a constant $\Delta t$ is assumed, which is only correct in a particular case, usually the average case. Hence, the calculation of the differentiation function contains inaccuracies, which manifests itself as noise, that can cause instability. In the continuous domain, the function can be implemented with zero latency using a simple electrical circuit, for instance using a single capacitor. The introduction of latency means the stability of signals can be affected by the latency introduced when moving from the continuous to discrete domain.

A great deal of work [21, 23, 24] has been performed analysing the effects of time varying delays on control system's behaviour, including the effect on stability of latency, jitter and data-loss. Considering the earlier example of the differentiation function, a missed sample or latency has a similar effect to jitter in that the actual value of $\Delta t$ is different to the value used in the calculations. The basic effect is that as these factors increase, the stability of the control system decreases. Torngren's work confirms that the management of these factors is fundamental to controlling system performance. Therefore, it is important that the approach to scheduling caters for the factors discussed in this section.

In our experience control systems are designed based on requirements that assume actions occur at pre-defined times, such as those found in purely hardware systems. However, in practice the effect of jitter is that actions do not occur systematically at these times.

### Scheduling of Control Systems

Even the most advanced scheduling and timing analysis approaches can only provide a statistical estimate of when functionality is performed. Functionality is performed at irregular times because the computation times of the software invariably changes due to the vast number of paths that can be navigated. Any change in the time at which events occur has an effect on the control algorithms. Torngren [16] identifies a number of properties (period, deadline and jitter) to be specified. The properties are related to both tasks and transactions. The aim of the properties is to provide control over the execution of software in order to provide an upper limit on the possible latency. The period defines the average time between updates. The period is defined in order to provide the required responsiveness of the system to the stimuli. In addition, regular updates of actuators is often used to correct mechanical drift. The deadline defines the maximum delay between a task being ready for release and the task completing.

Torngren, amongst others, makes the observation that good engineering practice implies that task rates, wherever possible, should be maintained as regular multiples of each other. Also, the iteration rates of transactions and the tasks that form them should be the same, where possible. By adopting these tactics, the

system should function more effectively, i.e. scheduling overheads are reduced, resource usage minimised and data flow simplified.

### 2.1.2 General Avionics System

A report on the General Avionics System [17] contains a specification for the general functions of such systems. The functions are data interactions and timing constraints for a mission control system typical of those found in existing U.S. Navy/Marine Corps aircraft. It should be noted that the system described is distributed, i.e. not all the functionality exists on a single processor. The specification for the system is significantly simpler than the one discussed by Torngren reviewed in section 2.1.1, since transactions are not considered. The requirements for tasks are a subset of those discussed in section 2.1.1. Therefore, no extra knowledge of timing requirements is obtained from that found in Torngren's paper. However, the General Avionic System's specification includes other requirements, such as the importance of a particular requirement being met and moding considerations. Also, the General Avionics System's specification confirms the requirements described by Torngren are accepted in other applications.

The importance of a requirement reflects the potential need to have multiple integrity levels on the same processor. In the simplest case, requirements could have two levels of importance; safety critical and non-safety critical. With respect to timing, the importance can be considered as either having to be met (i.e. hard deadlines) or should be met (i.e. soft or firm deadlines).

Moding is where particular software is only executed when the system has a certain status. Moding is often used in practical systems because it is not always necessary or safe to execute all the software all the time. For example, the software for operating the landing gear of an aircraft is not needed during flight.

### 2.1.3 Magneto Stereotaxis System

Wika and Knight's work [18] examines how the production of a safety kernel may be the easiest way to achieve or enhance the integrity of a particular system. A

safety kernel is a kernel that schedules the usual functionality, as well as the enforcing of safety policies. A kernel is the software that provides the fundamental system services such as scheduling, fault tolerance and handling hardware devices[1]. A safety policy is a requirement used to control particular functionality, normally for the purpose of policing certain hazards. The original concept of the safety kernel was introduced by Rushby [25].

Rushby states the benefit of the safety kernel philosophy, rather than embedding the functionality into the application, is that the kernel is small enough to be produced to a much more rigorous standard, for example using formal methods. The philosophy allows a powerful argument that can be used during the certification process, which formal techniques have been applied in key areas to reduce the likelihood of hazards. This is particularly important for systems to be certified against DEF STAN 00-55 [26], which advocates the use of formal methods throughout the design lifecycle of systems. In many papers, an example of which is [27], the opinion is stated that formal methods are impractical in real systems because they do not scale well to large systems, leading to high costs. As a compromise, the use of formal methods in key areas is seen as a sufficient fulfillment of requirements.

The Magneto Stereotaxis system was analysed by Wika to determine the safety policies to be enforced. A Magneto Stereotaxis system is a device that steers very powerful magnets during the process of detecting tumours in the brain. The incorrect control of the magnets could harm, or even kill the patient. This has lead to a number of safety policies derived through design, referred to as design-derived requirements. There could be a number of design-derived requirements aimed at making the kernel "police the system" operation in order to prevent any unsafe actions. For example, the X-ray source must be turned *off* for 0.2 seconds before an *on* command is executed. A separation requirement (e.g. task A is to be separated from task B by 0.2 seconds) is defined to ensure a time gap between the task performing the *on* operation, and the task performing the *off* operation.

Wika's work highlights a number of safety-related requirements, such as the need to protect against failures. Wika's work provides further support for the existing

[1]The terms kernel and scheduler are both used through this thesis as a term for the mechanism that controls task execution.

types of requirements raised through the examples of the DICOSMOS project and the General Avionic Systems.

### 2.1.4 Summary of the Timing Requirements

An evaluation of industrial practice in the avionics area confirms the requirements discussed in this section are realistic. This statement is supported by the case study presented in Chapter 7. There are also a number of other examples that have similar types of requirements to the ones discussed, including the Olympus Attitude and Orbital Control System [28] and the mine pump system [29].

There are four principal categories of timing requirements considered; those associated with tasks (i.e. existing on only one processor), those involving messages which transfer data between processors (i.e. existing on the databus), those associated with transactions (i.e. these could exist on more than one processor, and thus include both tasks and messages), and those that are derived (i.e. related to the system design).

The timing requirements for a task are:

1. *Period* - All tasks can be considered to have a period. Sporadic tasks are modelled as a periodic task whose period is equal to the sporadic task's minimum inter-arrival time. A sporadic task is one where the task is released not at a regular rate, but instead following a certain event occuring.

2. *Deadline* - The deadline of a task is the maximum time allowed from the expected task release until the completion of the task execution.

   The importance of the timing requirement is normally categorised by whether the deadline is hard or soft. Hard real-time requirements are those that must be met in all cases. Soft real-time requirements are those that should be met in the majority of cases, but it does not matter if the occasional deadline is missed. Locke [30] introduces a third category of *firm* deadlines that should be met in the majority of cases. A firm deadline is similar to a soft deadline except if a firm deadline is missed, then the task should not be completed. Whereas, a task with a soft deadline is completed anyway.

In safety critical systems most requirements are considered to be hard real-time. One reason is the difficulty in assessing whether soft requirements can affect hard requirements. Soft requirements would tend to be implemented to a lower integrity standard than hard requirements, which means they may be less predictable. Therefore, it is often easier to assume all tasks have hard deadlines.

3. *Jitter* - The jitter constraint for a task is the allowed variation of task completion from precise periodicity. Jitter is generally caused by variations in the worst-case execution time of tasks. Jitter constraints are normally placed on the outputs from the system to ensure the occurrence of actions does not vary too much.

A sequence of tasks executed in a fixed order is referred to as a transaction. The timing requirements for a transaction are:

1. *Period* - Similar to a task, a transaction has a periodic requirement. It is not unusual for some of the tasks in the transaction to be executed at different rates. In these cases, the transaction period is equal to the least common multiple of the tasks' (that form the transaction) period. The reason the least common mutiple is chosen, rather than a lower value, is that the tasks can only execute in the required precedence order this often.

2. *End-to-End Deadline* - Transactions normally have a requirement that all tasks are executed in a particular order within a given amount of time. Again, transaction deadlines can fall into the same categories as for tasks: hard, firm and soft.

3. *Jitter* - Similar to a task, a transaction may have a jitter constraint. However in general, a jitter constraint is only applied to the tasks in the transaction that gather the inputs or produce the outputs. These are the tasks where jitter has most effect, refer to section 2.1.1 for more details.

To implement a transaction, tasks are executed in a fixed order on a number (possibly just one) of processors. For transactions featuring tasks on different processors, a message is scheduled on the communications system to transfer data. For example, if the system illustrated in Figure 2.1 is implemented as part

of a distributed control system, then the functionality for the sensor, actuator and transformations could be executed on separate processors. The reason this might be necessary is so that the processing can be placed where it is physically needed to reduce the amount of cabling. Messages are then used to communicate the required data between the sensor and the processor performing the transformation, and then another message to communicate the data to the actuator. Therefore, the message will have the following design-derived requirements:

1. *Period* - The message has a periodic requirement equivalent to the period of the task that sends it.

2. *Deadline* - The deadline of a message is the maximum time allowed from the earliest message release time until the time when the receiving task is due to be released.

In addition, there will be a number of requirements that could be obtained from other sources, predominantly as a by-product of the design process. Derived requirements related to the scheduler include:

1. *Precedence* - A precedence requirement may be specified to ensure that a particular set of actions always occur in the required order.

2. *Separation* - A particular sequence of events may have a minimum time separation enforced to ensure correct operation. Separation requirements may be used to provide controlled access to a different device from different tasks, i.e. to not allow resource contentions to occur.

Any approach derived for scheduling should be able to provide proof these requirements are met. A need for any scheduling policy is to have an approach to task attribute assignment that effectively deals with the requirements.

## 2.2 The Influence of Certification Requirements on Scheduling of the System

One aspect that makes the work in this thesis different to other work performed on scheduling is that it is targetted at a specific domain that demands certain guarantees to be made when justifying system integrity. The purpose of this section is to help understand the constraints imposed by the certification standards on the scheduler and the timing aspects of the system.

There are a multitude of certification requirements for safety critical systems. An accurate and relevant observation by Tannenbaum [31] is "*The nice thing about standards is that there are so many of them to choose from*". The work in this thesis is to be based on the United Kingdom's certification standards for the production of avionic systems. The standards include: the military standards DEF STAN 00-55 [26] and 00-56 [32], and the internationally agreed civil standard DO-178B [33]. The standards are considered at a relatively high level in order to abstract away from specific characteristics and hence capture the key points.

The consideration of the influence of the standards may be separated into a number of parts, which includes: the timing requirements to be met, the schedule and scheduler implementation, and the verification techniques.

### 2.2.1 Timing Requirements From the Certification Standards

The purpose of this section is to explore how the certification standards implicitly lead to timing requirements on the system. The requirements mostly relate to the ability to tolerate faults. For example section 5.2.2 of DO178B [33], "*Responses to failure conditions should be consistent with the safety related requirements*". The aim of the requirements is to provide control over how faults are dealt with.

Requirements that deal with faults can be derived from a reliability measure of how long the system can be "at risk", i.e. have one replicated version of a component unavailable. In many systems encountered, the requirement is that faults are identified, tolerated and recovered (where possible) in bounded time.

A modern aircraft, particularly military ones, tend to be unstable inorder to improve manoeuvrability. Therefore the aircraft relies on computers to assist the pilot. The X31 aircraft is an example of a system where a failure of the flight control system for more than a few milli-seconds is catastrophic.

The timing requirements for fault tolerance may be dictated by the need to:

1. still meet timing requirements in the event of certain failures, or
2. restrict the time at risk when parts of the system are unavailable, or
3. limit the time for which the entire system is unavailable without a catastrophic event happening.

From a timing perspective, failures tend to manifest themselves as timing overruns. Any approach to scheduling must provide adequate control of timing overruns commensurate with the integrity of the system.

### 2.2.2 The Influence of the Certification Standards on the Scheduler Implementation

Similar to the observation in section 2.2.1, there are no explicit requirements in the certification standards for producing software that affect the scheduler implementation, but there are implicit ones. The implicit requirements are related to how certain architectural features are more difficult to verify. Two specific influences, both related to software execution, have been established through contact with companies developing safety critical systems.

The first influence is related to the flow control provided by the scheduler, where the arbitrary interrupting of the application's tasks is not advisable because of the increased difficulty in attaining sufficient test coverage. Arbitrary interruptions lead to a vast increase in the potential paths within the software when compared to code with no interruptions. The certification standards do not prohibit the use of interrupts, but simply highlight the potential difficulties in guaranteeing system integrity when they are used.

The second influence concerns multiple levels of interrupt, which raises issues as to whether the interrupt handler functions correctly. Correct functionality includes:

1. when the interrupt has been handled returning execution to the point at which the interrupt occurred,
2. the interrupt must only disrupt execution when the current execution is at a lower integrity level (including all applications correctly relinquishing control), and
3. the interrupts should never be nested.

Nested interrupts are when interrupts occur while other interrupts are still being handled. The certification authorities tend to frown upon the use of interrupts because of the difficulty in guaranteeing they function correctly in relation to the rest of the system. Normally significant attention is paid to the relatively simple problem of the interrupt that signifies the start of a minor cycle with the cyclic scheduler. Multiple levels of interrupt would cause a significant increase in the effort required to certify the system.

The two influences are considered to be implicit in the certification standards through the discussion of testing. For example, section 37.1.1 of DEF STAN 00-55 [26] states *"..Sufficient testing should be performed to show that the dynamic and performance requirements have been met, and that the assumptions used by formal methods are valid for the target system"*. Clearly to achieve the goal of full test coverage is easier in a system with no interrupts than in a system with arbitrary interrupts. Except for the most trivial example, complete test coverage is considered impossible. However, the system not having interrupts allows testing to come closer to achieving complete coverage than if the system used interrupts.

Even though the two influences on scheduler implementation are implicit, they used to be explicitly stated in an older version of Interim Defence Standard 00-55 [34]. Their removal could be a reflection on either the need to allow systems to become more advanced or a desire not to have unnecessary design constraints. Any approach to scheduling should minimise the amount of interrupts that occur, and to ensure the ones that do occur have predictable outcomes.

### 2.2.3 The Influence of the Certification Standards on the Verification of Software

A key part of the certification requirements is the issue of how verification is performed. Verification is to provide evidence that the system meets its requirements. The requirements cover a number of areas all of which fall into two categories; functional (how inputs are translated into outputs) and non-functional (properties related to the infrastructure such as timing, memory, communications and safety). The influence on timing requirements is already dealt with in section 2.2.1.

The certification standards tend to advocate that verification is performed using analysis, and testing is used to ensure that the requirements are sufficient and correct as well as to provide extra verification evidence. Analysis is also used to ensure the requirements are consistent, i.e. ambiguities do not exist. Where analysis is not feasible, then comprehensive testing is mandated. In addition, traceability is required through all parts of the system and documentation in order to show where and how each requirement is achieved.

There are many more requirements advocated by the certification standards. For an in-depth survey of certification requirements, refer to Papadopoulos and McDermid [35]. A key contribution of Papadopoulos' paper is a consideration of the commonalities and differences between standards. Based on the certification standards and the work of Papadopoulos, the common elements related to scheduling are:

1. The argument of safety or dependability is based on procedural and technical evidence. This helps justify the benefit of timing analysis compared to test as part of the verification process because the evidence is precise.

2. A safety process that drives the system development process. This implies timing analysis should be a fundamental part of the overall lifecycle, rather than being performed after the event.

3. Safety requirements are established from functional hazard analysis and risk assessment. This relates to the need for safety policies such as the requirement to detect, tolerate and recover from failures within bounded time.

## 2.3 The Current Lifecycle for Producing a Schedule and Scheduler

Any technology transfer exercise has many effects on the way companies operate. To understand the impact of the new technique, such as a change of scheduling policy, it is necessary to assess how the current process has to be modified to accommodate the change and consequently how change provides benefit. Without this understanding, it is unlikely that the changes would be accepted. This section tries to define the existing process model at a sufficient level of detail for the assessment of the effect of change to take place. There are two facets to the discussion, which are the production of an appropriate scheduler and the production of an appropriate schedule. In our experience the processes used in industry are difficult to define (because the process is complex and each individual working on the project could have their own slightly different way of working) and are prone to change (because the process for a project is often negotiated with the customer). Many of the statements contained in this section are based on the process used by Rolls-Royce in the design of aeroengine controllers [36]. However, the description is considered to be equally applicable to many other processes.

Any change of scheduling regime should consider the attributes of the current process and attempt to "trade-off" the impact of the change on the process against the benefit that is obtained. There is a fine balancing act when judging the changes that may be beneficial; on the one hand radical change could lead to significant benefit for scheduling, but on the other hand radical change could have greater system-wide impact and hence increased risk. Also, changes to the scheduler need to be considered within the wider scope of the system.

There are a number of stages of the lifecycle not mentioned, such as configuration management, which are important but not particularly to scheduling. The lifecycle stages relevant to the scheduling of the system are considered to be: requirements; design; implementation; verification and certification. There is a great deal of literature [37, 38, 39] that investigates the software engineering lifecycle in detail. However, a general definition has been produced here in order to allow the impact of change to be assessed. Clearly, the process defined can not

be a perfect match for all lifecycles used to develop systems but it is considered to be a sufficient representation.

### 2.3.1 Requirement Specification

Requirements specification is the mechanism of capturing and specifying what the system is to do, in terms of functional and non-functional requirements. The purpose of the requirements is to convey to the system design team what the implementation is to do for both design and verification purposes. A key need is to show that the requirements are consistent, complete and correct. The scheduling requirements tend to be expressed in a natural English form, i.e. free flowing sentences are used to express the requirements.

Based on experience, the timing requirements are a legacy from the original hydro-mechanical systems. A key issue is that iteration rates may have been chosen with prior knowledge of the available iteration rates with the cyclic scheduler, leading to "practical" rather than "actual" requirements. Significant benefits could be obtained by addressing what are the real timing requirements, not least because a criterion for successful technology transfer is sufficiency.

### 2.3.2 Design

Design is the practice of determining how the requirements should be implemented. A key part of any design is justifying and demonstrating how the requirements are met, i.e. which parts of the design relate to which requirements.

With respect to the schedule, informal techniques are generally used for design. The cyclic schedule is initially synthesised for the first build using a tool that places tasks in a number of minor cycles so that the scheduling requirements are met. The synthesis is performed using estimated values for the worst-case execution times of tasks. The schedule is sometimes never re-synthesised. Instead, when there is a requirements change or a cycle overrun, the schedule list is manually altered. This approach is adopted so the effect on the data and control flow of the system is easier to judge, allowing a reduction in the level of regression testing. If the schedule is re-synthesised, then a complete re-verification of the system is required and all the built-up safety evidence is lost. This represents

a weakness in relying on having a static run-time ordering as provided by the cyclic scheduler. The weakness is that although such an ordering may be easier to guarantee because it is deterministic, when the ordering is changed the whole basis of the guarantee is lost.

### 2.3.3 Implementation

Implementation is the mechanism of building components to meet the requirements with sufficient integrity. A key part of this stage is justifying and demonstrating that the eventual implementation meets the software specification and design.

Many of the United Kingdon avionics industry currently implement their systems using the SPARK (SPade Ada Real-time Kernel) Ada subset and SPARK annotations [40]. The SPARK subset eliminates parts of the language whose implementation characteristics are difficult to determine. Hence, the likelihood of safe software is increased. The SPARK annotations support the information required to perform static analysis. However, the key point is that SPARK Ada intends to introduce a philosophy that is used throughout the design to add benefit to the resultant system. The basis for the SPARK technique is that it promotes good software engineering because it encourages the designer to reason about the design as it is produced.

The benefit of using a subset of a language, such as SPARK Ada, is that then the operation of the code is predictable. This means that worst-case execution time analysis can be built to model the system's timing behaviour.

### 2.3.4 Verification

Verification is the mechanism of ensuring the system meets the requirements. In terms of cost and time the verification stage is the most expensive for systems developed to safety critical standards [2]. Testing of the system's timing properties involves measuring the worst-case execution time and the period of tasks, followed by analysing whether the timing requirements are met. The worst-case execution times of tasks are measured during the simulation of a vast range of realistic scenarios. The time taken to measure the worst-case execution times

needs to be commensurately large to provide a sufficient degree of confidence that the measured value is close to the actual worst-case. The tasks' worst-case execution times are then used to prove the system meets its timing requirements.

To test the system, scripts are derived from the requirements that establish whether the sub-system operates correctly on its own and as part of a larger system. There are two problems with a test-orientated approach, which are; the time taken to perform comprehensive tests is large, and the values obtained are almost certainly optimistic. The value is optimistic since it is unlikely the worst-case path is exercised (except in the simplest of cases) due to the large number of possible paths. Optimistic values may occur especially when some of the paths are rarely exercised, for example paths that deal with failures. Unexercised paths lead to uncertainty in the analysis to show requirements are met, which means cycle overruns could occur when the system is in service. A cycle overrun is when the execution of tasks in a minor cycle extends into the next minor cycle.

The reason analysis techniques are rarely used for establishing a task's worst-case execution time is because the results tend to be overly pessimistic. The pessimism may be such that the verification results imply the system's timing requirements are not met when they actually are. There is a vast body of research that attempts to solve the problem of pessimism in worst-case execution time analysis, the following are examples [41, 42, 43]. However, few of the techniques have been used in industrial systems. The reason is the techniques rely on support from the compiler and processor vendors (i.e. the supplier providing necessary information on the product and the product providing output of an appropriate form), which is not currently available. Another problem with worst-case execution analysis is that it is rarely portable between different targets, leading to a different version being required for each.

The timing analysis contained in this work assumes actual worst-case execution times are available, so the problem becomes a matter of showing whether timing requirements are met.

### 2.3.5 Certification

Certification is the mechanism for arguing the system meets its safety obligations, which are derived from the knowledge of the risk (severity and likelihood) of

hazards. Throughout the development process, specific evidence needs to be captured that allows the safety of the system to be argued. The safety argument is presented in the certification case. The certification case should evolve through the development life-time of the system.

A simplified view of the current approach towards certification is to use a number of well known techniques [44] (for example fault tree analysis, which is a technique used to determine how a particular fault could occur) as well as other information obtained from other stages of the lifecycle such as the results of timing analysis. However, whatever techniques are used, a strategy is still required to argue the system is safe.

There are two basic strategies used that are applied on their own or as combined evidence to argue the system is safe. The strategies are:

1. *Software Safety Analysis* - Use analysis techniques to show the system is of the appropriate integrity. The analysis justifies the four previous stages of the lifecycle (Requirements Specification, Design, Implementation and Verification) leads to correct, complete and consistent requirements. These requirements are refined into a system in a predictable manner.

   Few projects perform software safety analysis due to the difficulty of the task and the immaturity of the available techniques. Instead, the software is produced to what is deemed an appropriate process. Software safety analysis could be considered relatively immature compared to other areas of engineering. For instance, there is no recognised technique for evaluating software reliability even though it is needed to justify the overall system reliability. During the reliability calculations of the system only hardware reliability is accounted for, as software reliability is assumed to be ideal (i.e. probability of failure is zero).

2. *Infrastructure Safety Analysis* - A different approach is to produce the software safety analysis as if the applications based on the system are of a lower integrity and concentrate on the correctness of the infrastructure. Therefore, the amount of software safety analysis that needs to be performed is reduced. This is a similar philosophy as for safety kernels discussed in section 2.1.3, where it is deemed impractical to produce all the software formally. The safety argument becomes a justification that the system is of

a reasonable quality, and the infrastructure manages the integrity. The approach requires the infrastructure to implement any safety policies derived from the system hazard assessment exercise.

It is unlikely a single technique on its own is sufficient to guarantee integrity. Independent of whether a software safety analysis is performed, the safety case would need a justification that the infrastructure detects and tolerates failures in a manner consistent with the needs of the system. In our experience, many systems tend to use a combination of both approaches to justify the integrity of the system. In particular, the scheduler is considered a key part of the infrastructure of the system. A great deal of effort is applied to ensuring tasks are released as expected and that timing overruns are detected correctly and in a timely fashion. Whereas, the actual schedule is considered to be slightly less important.

## 2.4 An Infrastructure For Cyclic Scheduling

The introduction of any new technique has many effects. For scheduling, a major impact is that the infrastructure of the system may need to be altered for efficient and effective operation. To understand any such transition, the typical architecture for a cyclic scheduler is outlined. Again, the discussion in this section cannot possibly represent every form of system produced, instead it is meant to be a representative example.

Typically the infrastructure for the cyclic scheduler generates a clock tick (in the form of an interrupt) that is used to signify the start of a minor cycle. The tasks are then released in a pre-defined order. A key feature of the task release mechanism is that tasks, once released, are executed to completion. This is referred to as non-preemptive execution. When the main software in the minor cycle has finished executing, control is passed to a background scheduler. The background scheduler executes less important functions until another clock tick arrives to signify the start of the next minor cycle. Therefore, the software to be scheduled in the background is implemented so that arbitrary interrupts do not affect system integrity.

Most implementations take advantage of the clock tick being available to initiate the timing watchdog functionality. The purpose of the timing watchdog is to detect and tolerate timing overruns. A frequently taken strategy for the timing watchdog is to have two modes of operation for executing tasks. The main scheduler is executed in "supervisor" mode, i.e. at a higher interrupt level than the background scheduler that is executed in "user" mode. If the system is operating in a timely fashion, then the tasks executing in "supervisor" mode should always be finished before the next clock tick. Therefore, the timing watchdog simply detects the mode the system is in when the clock tick arrives. Supervisor mode means that a timing overrun has occurred, whereas user mode means no overrun has occurred.

The supervisor/user mode is normally observable by an electrical signal from the processor. Therefore, a simple timing watchdog may be produced in hardware, which is triggered when there is a clock tick. The timing watchdog hardware checks the mode of the processor, and fault recovery is performed if the processor is still in supervisor mode. The typical fault recovery strategy is to switch the lane controlling the system to a replicated version and then reset the lane in which the fault exists. This assumes another lane is available, and this lane would probably be actively redundant. The timing watchdog approach is considered effective for two reasons, which are; there is no software involved in the test and the functionality is simple.

## 2.5 Problems With The Cyclic Scheduler Approach

During projects to develop safety critical systems, there are often problems related to timing. The problems include; resource budgets are exceeded, timing requirements are not met, unexpected failures arise in service, and resources are wasted. The fact that projects frequently experience problems with timing is related to a number of widely recognised weaknesses in the cyclic scheduling model, documented by Locke [6]. The weaknesses include inflexibility, poor maintainability, and lack of support for future needs. The weaknesses are considered in the following sub-sections.

### 2.5.1 Inflexibility Leading to Wasted Resources

The inflexibility of the cyclic schedule has two restrictions. These are a limited amount of iteration rates are available and tasks have to be periodic. The inflexibility often results in requirements being expressed to suit the model, i.e. only periodic tasks that are harmonics of the minor cycle rate. The inflexibility leads to wasted resource (when the worst-case scenario is considered) caused by:

1. Tasks being executed faster than necessary to meet its period requirement. For example, to implement a task with a rate of 40 units in a cyclic scheduler that has a minor cycle rate of 25 means the task must be queued in every minor cycle, i.e. at a rate of 25. Queuing the task every other cycle would result in an average rate of 50, and the requirement is not met. The resulting waste in resources is $\frac{15}{25} \times 100 = 60\%$.

2. Sporadic tasks having to be implemented using a polling periodic. A polling periodic is a task that executes at a regular rate to check whether a particular event has occurred. Using a polling periodic, the average case may have similar performance but the worst-case would be pessimistic. For example, a sporadic task with a minimum inter-arrival rate of 25 but an average rate of 1000 would have to be implemented using a periodic task with a rate of 25. However if the average rate is 1000, then there is a $4000\% = (\frac{1000}{25} * 100)$ waste in resources.

   An alternative example is a sporadic task with a minimum inter-arrival rate of 1000 but a deadline of 50 would have to be implemented using a periodic task with a rate and deadline of 25. The reason the periodic task does not have a period and deadline of 50 is that the task would suffer a release jitter equal to its own period due to uncertainty of when the event occurs. Again, there is a 4000% waste of resources.

Matters are made even worse by the very cyclic nature of the schedule. Tasks have to be placed in each minor cycle so that the cycles do not overflow. To prevent overflows, space is reserved at the end of each cycle to act as a "timing margin". Any type of schedule is invariably produced with a timing margin but the cyclic scheduler has a number of these (one for each minor cycle) that makes matters worse.

Overall, a significant proportion of the valuable resources available may be wasted. The waste is especially significant when it is considered most safety critical systems use older generation processors for the reasons explained in section 1.1. Invariably, there is always a shortage of processing resource. Recapturing this resource is a major driver for the research reported in this thesis.

### 2.5.2 Maintenance Difficulties with the Cyclic Scheduler

The maintenance difficulties of the cyclic schedule arises from the need to correctly order tasks so that both task and transaction requirements are met. In a heavily loaded system each minor cycle is nearly full. As the execution times of tasks change there is a frequent need to move tasks between minor cycles to balance the load and prevent overflows. In itself, this can lead to some complicated juggling acts to "fit everything in".

In most cases, the maintenance difficulties of the scheduler is enough to convince people that a more appropriate solution is necessary. Garey and Johnson [45] show the bin packing problem, which is equivalent to the synthesis and maintenance of the schedule, to be NP-complete. However, the problem is made even more difficult when transaction requirements have to be met. The introduction of transactions means the job of moving tasks is further complicated by the interactions between particular tasks that have to be maintained. Simply moving tasks to balance the minor cycles, without consideration of the transactions, could break precedence constraints previously met.

The computational complexity of synthesising the schedule suggests that tool automation should be used. Many different techniques have been proposed to the problem of synthesising cyclic schedules. Burns et al [46] performed an analysis of some of the available techniques for reducing the complexity of the problem. Their paper looks at five different techniques (simulated annealing, brute force, genetic algorithms, heuristic search and stochastic evaluation) for synthesising the schedule. Burns et al concludes each technique has its own benefits and drawbacks, and the technique chosen depends on the specific needs of the system.

Automated synthesis may result in a better scheduling solution, but at what cost? In many cases tool automation is the solution adopted. However, safety critical

systems introduce their own specific problems. The main problem with using automated tools is related to regression testing. It has already been stated that verification is the largest cost in projects to develop safety critical systems. Using a tool for synthesis gives a new schedule where all the task interactions may be different. This leads to a significant effort being required during re-verification. A commonly recognised characteristic of safety critical systems is that during their development the systems are prone to a large number of changes [2]. Using an automated tool would vastly increase the cost of verification due to the amount of regression testing needed on the functional requirements. Therefore, an approach often taken is to attempt to manually modify the schedule, which is difficult and may lead to requirements not being met. However, the effect of change is now easier to assess.

A general system maintenance problem introduced by the cyclic schedule is summarised by Sha, Liu and Goodenough [47], "*Under the cyclic approach, meeting the responsiveness, schedulability and stability requirements has become such a difficult job that practitioners often sacrifice program structure to fit the code into the right time slots*". The system used as a case study in Chapter 7 has software split into more than seventy tasks (based on the need to have small enough chunks of software to organise a cyclic schedule), rather than having less than ten tasks (that represents the principal functional partitions of the system). There are a number of side effects of splitting tasks, including; there are more tasks to be scheduled and verified. Also, the software may no longer have the decomposition dictated by good software engineering practice.

### 2.5.3 Supporting Future Requirements

It is likely that system designers are going to place more and more functionality into the software, and there is a trend to move away from uniprocessor systems. The move towards distributed systems is likely to expose further weaknesses in the cyclic scheduler.

The maintenance of cyclic schedules involving one processor is difficult enough, but many processors all working together would be even more complicated. For instance, the introduction of distributed transactions results in a number of schedules cooperating to meet a common requirement. Carlow [48] presents an

example of a well-known timing problem related to the difficulty in handling distributed systems with the cyclic schedule. Carlow describes how the first launch of the Space Shuttle Columbia was delayed due to a failure to synchronise two computers of a distributed system.

An implication of distributed transactions is that a change to the schedule on one processor has a knock-on effect across all the schedules. This greatly increases the computational complexity, therefore maintenance costs are multiplied. The problem is further exaggerated by the recent trend towards multi-company and even worse multi-national collaborations, where the need to keep changes localised is large. It may be prohibitive for systems to be designed in a manner involving tight cohesion between schedules on different processors. However, the use of cyclic schedules to implement distributed transactions would necessitate the cooperation of all the schedules.

Industry recognises the problems of the cyclic schedules within the systems currently being produced and maintained, and the fact they are likely to be exaggerated in the future. Therefore, a change of scheduling approach is required particularly if future needs are to be supported.

## 2.6 Summary

In this chapter the principal subject that has been addressed is identifying how systems are currently developed. The subject has been addressed in a relatively generic manner so that the transitions discussed within the thesis are widely applicable. The purpose of identifying how systems are currently developed is so that a baseline is identified for the transitions to start from. The baseline has been defined for four main areas; life-cycle, infrastructure, typical timing requirements and the certification requirements. In summary, the characteristics of current systems includes:

1. periodic tasks that are executed non-preemptively,
2. some tasks execute in an integrated fashion as transactions,
3. the control of some tasks' jitter is critical to the correct operation of the system, and

4. there is a single regular interrupt to facilitate the task release mechanism and timing watchdog.

The problems with cyclic scheduling were then discussed. Section 1.2 suggests that fixed priority scheduling is the likeliest solution to the problems of cyclic scheduling. The remainder of the thesis will explore this claim.

The literature survey of Chapter 3 provides details of work performed on fixed priority scheduling. Chapter 4-6 uses the information in the literature survey and the details of this chapter to derive a suitable approach for using fixed priority scheduling in safety critical systems based on a single processor. To manage the transition, the criteria for successful technology transfer (certification, sufficiency, understanding and reuse) should be considered when making design decisions. Chapters 4-6 identify existing theory that can be used with or without modification. The chapters also develop new theory to fill the necessary gaps, and providing an argument for the techniques' adoption. The findings of Chapters 4-6 are demonstrated in the case study of Chapter 7.

# Chapter 3

# Literature Survey for the Fixed Priority Scheduling Technique

Chapter 1 highlights the fact that fixed priority scheduling may provide solutions to many of the problems encountered when using cyclic scheduling in safety critical systems. The purpose of this chapter is to investigate the literature for fixed priority scheduling in order to understand the existing work on the subject. There are a number of areas to be surveyed to justify the two transitions (cyclic scheduling to fixed priority scheduling and uniprocessor to distributed systems), and the actual technical work contained within the transitions.

In section 3.1 the computational model, priority assignment and timing analysis for fixed priority scheduling are introduced. In section 3.2 a survey is presented to support the other transition of interest, which is from uniprocessor to distributed systems. Many new issues, such as meeting distributed transaction requirements are introduced, for which solutions are to be investigated. In section 3.3, the MARS (now referred to as Time Triggered Architecture) approach developed by the University of Vienna is discussed. The MARS approach is deemed to be the best of the currently available public domain approaches for safety critical systems. Finally, section 3.4 looks at work on Integrated Modular Avionic (IMA) systems that represents a proposed future architecture for avionic systems.

## 3.1 Fixed Priority Scheduling on Uniprocessor Systems

The purpose of this section is to present a review of the literature on fixed priority scheduling for uniprocessor systems.

### 3.1.1 Early Work on Fixed Priority Scheduling

The early work on fixed priority scheduling is published by Liu and Layland [10]. In their paper they discuss fixed priority scheduling within a simple conceptual framework, where the task executing is always the highest priority task that is runnable. Therefore, the tasks execute in a preemptive fashion. In Liu and Layland's model, tasks are assigned priorities so that the highest priority task has the shortest period, referred to as rate monotonic scheduling. Liu and Layland shows the rate monotonic scheduling model is optimal for task sets with zero offsets and deadline are equal to period. An offset is classed as the time a task release is separated from a common reference of time. Typically, the time reference coincides with when the majority of tasks are released. Liu and Layland, and Serlin [49] propose the schedule can be verified using a simple utilisation-based test. The test is expressed in equation (3.1).

$$U_{max} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{3.1}$$

where $n$ is the number of tasks

$i$ is a task in the set of tasks

$C_i$ is the worst-case execution of task $i$

$T_i$ is the period of task $i$

$U_{max}$ is the maximum processor utilisation

Equation (3.1) indicates that if the utilisation test is less than a value (dependent on the number of tasks in the system), then the system is schedulable. Using equation (3.1), if $n$ is equal to one and $U_{max}$ is less than or equal to one hundred percent then the task set is schedulable. As $n$ tends to infinity then $U_{max}$ has to

be less than or equal to 69.31% for the task set to be schedulable. The utilisation-based test provides pessimistic results. An example of the pessimism is a task set with two tasks whose period and deadline equal to $T$, and worst-case execution times of $T/2$. The utilisation for the two tasks is 100% so the test implies the task set is unschedulable since, based on equation (3.1), $U_{max} = 82.82\%$. However, the task set is schedulable because the tasks are simultaneously released, one task would be dispatched immediately and execute for time $T/2$ and then the other would be dispatched and execute until time $T$. Therefore, both tasks execute within their deadline.

Later work by Lehoczky, Sha and Ding [50] shows how for typical task sets with a large number of tasks, utilisations approaching 90% are schedulable. This is despite the utilisation based test indicating the task set would be unschedulable.

Liu and Layland also identified the "critical instant", which is the point in time when all tasks are simultaneously released. A task instance is defined as the release, execution and completion of a task. The key results related to the critical instant are that:

1. if all tasks execute for their worst-case time and all tasks are subsequently released at their maximum rate, then tasks have their worst response;

2. the timing behaviour is monotonic - monotonic is where there is an identifiable worst-case scenario that if the timing requirements are guaranteed as met in this case, then they are always met, and

3. the worst-case response time of every task is the response time of the task instance released at the critical instant. Therefore, only one instance of each task needs to be demonstrated as meeting its timing requirements.

Liu and Layland's observation about the critical instant and monotonic behaviour is significant because the subsequent timing analysis can be simplified to ease computational complexity as well as helping to meet the understanding criterion by the use of relatively simple, well-understood and accepted analysis.

After the initial work on fixed priority scheduling only a small amount of work was published [51]. However, during the late 1980's fixed priority scheduling began to attract more attention and publications were written presumably as

the complexity of systems started to demand greater flexibility and efficiency. Within avionic systems, this relates to the introduction of more computerised control into systems rather than hydro-mechanical control.

### 3.1.2 Analysis of Tasks' Worst-Case Response Time

A number of observations were made by Strosnider, Katcher and Arakawa [52], these include the utilisation-based test is pessimistic and the priority assignment algorithm can be sub-optimal. Rate monotonic assignment of priorities is sub-optimal if tasks' deadlines are not equal to their period or tasks have non-zero offsets. The pessimism in the utilisation-based test indicates a need for better timing analysis. There are three categories of timing analysis:

1. *Sufficient and Necessary* - The analysis always indicates a schedulable solution when the task set is schedulable. This category of timing analysis is most desirable.

2. *Sufficient and Not Necessary* - If the analysis indicates a schedulable solution, then the task set is schedulable. However, there are cases when the task set is schedulable contrary to the results of the analysis. This category of analysis is acceptable in cases where the *sufficient and necessary* analysis is considered infeasible, or is not available.

3. *Not Sufficient and Necessary* - The analysis indicates a schedulable solution when the task set is in fact not schedulable. This form of analysis is undesirable.

The original consideration of timing analysis, presented by Leung and Whitehead [53] shows that if every instance of every task on a particular processor is schedulable over the period [Maximum Offset of Any Task, 2×Least Common Multiple of the Task Periods + Maximum Offset of Any Task)[1] then the task set is schedulable. Audsley [54] improved the analysis so the duration became [Maximum Offset of Any Task, Least Common Multiple of the Task Periods +

[1]The brackets [.,.) indicate a range where the first value is included and the second value is excluded. For example, [0,8) refers to the time $t$ in the range $0 \leq t < 8$.

Maximum Offset of Any Task). Later work by Lehoczky's [55] presents busy period analysis, which provides further reductions in the duration to be analysed. This was achieved by only considering intervals when higher priority tasks are queued, i.e. those that can cause interference.

Leung's and Audsley's analysis, frequently referred to as exact analysis, definitely helps to reduce the pessimism of the utilisation based test and is classed as sufficient and necessary. The problem with the analysis is that for task sets featuring irregular periods, particularly co-primes, a result would be very difficult to obtain. This is because the value of least common multiple of task periods becomes large. A large value for the least common multiple means that a large number of task instances have to be checked.

### 3.1.3 Computationally Feasible Analysis for Tasks' Worst-Case Response Time

Harter [56, 57] presents the first work that attempts to solve the computational complexity problems of the exact analysis. Harter developed timing analysis that could be performed in pseudo-polynominal time to show whether the task set is schedulable, i.e. all tasks meet their deadline. The timing analysis for each individual processor is solved using equation (3.2) taken from [57]. The analysis is valid for task sets with a critical instant. Harter's analysis assumes all tasks have a fixed unique priority, zero offset, and the deadlines are not greater than the period.

$$R_i = C_i + I_i + B_i \tag{3.2}$$

where i is a task in the set of tasks for a given node

$R_i$ is the worst-case response time of task $i$

$C_i$ is the worst-case execution time of task $i$

$B_i$ is the blocking time of task $i$

$I_i$ is the interference of task $i$

The blocking time, $B_i$, is the longest time that a lower priority task can prevent the execution of the task being analysed when it is runnable. The blocking time is dependent on the computational model that is being used. In an idealised

preemptive model, the blocking time should be zero. However, cases exist particularly with shared resources, where some blocking may need to be accounted for.

The interference a task suffers is the maximum utilisation from the critical instant of its higher priority tasks before it executes for the first time. The interference is calculated using equation (3.3). Therefore, the interference is the sum of the utilisations over the duration of interest for all the higher priority tasks than task $i$. The utilisation is the product of the number of times the task can execute and its worst-case execution time. The number of times a higher priority task can execute is found by rounding up the result of the time during which interference may occur (i.e. the response time of the task being analysed) divided by the period of the higher priority task.

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{3.3}$$

where $hp(i)$ is the set of higher priority tasks than task $i$

Equation (3.2) is solved by forming a recurrence equation as shown in equation (3.4).

$$R_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{3.4}$$

with $R_i^0 = C_i$
which terminates when $R_i^{n+1} = R_i^n$, or $R_i^{n+1} > D_i$.
where $D_i$ is the deadline of task $i$

The analysis in this section was also derived and published independently by Joseph and Pandya as the Time Dilation Algorithm [58, 59], and by Audsley et al [60].

The worst-case response times calculated using Harter's equations are based on Liu and Layland's model where there is a critical instant and all tasks execute for their worst-case execution time, which is the worst-case situation. Audsley [54] shows how under certain conditions (all tasks have zero offset, tasks are executed preemptively, and deadlines are less than or equal to their period) the test is sufficient and necessary. The equations verify the system is schedulable in pseudo-polynomial time [61].

### 3.1.4 Analysis for Tasks with Release Jitter

When implementing practical systems the ideal timing analysis model (for example no overheads) in section 3.1.3 is not sufficient, which could lead to optimistic results. One of the areas the model is deficient is that it assumes tasks are always released at the correct time. In practice, tasks may suffer release jitter for a number of reasons, including there being an imperfect task release mechanisms. For example, a sporadic task modelled by a polling periodic task of rate $T$ would have release jitter of up to $T$ units of time. This accounts for the worst-case event arrival, i.e. just after the task has polled for the event. Tindell [62] expanded the analysis to account for the extra interference caused by release jitter by modifying equation (3.3) to form equation (3.5). The principal difference is that for equation (3.5) the duration in which interference can occur is augmented to allow for higher priority tasks being released later than ideal.

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \tag{3.5}$$

where $J_j$ is the release jitter of task $j$

When the analysis converges, $\mathrm{R}_i^{n+1} = \mathrm{R}_i^n$, the worst-case response time is expressed as:

$$R_i = R_i^{n+1} + J_i \tag{3.6}$$

### 3.1.5 Deadline Monotonic Priority Ordering

A limitation with rate monotonic scheduling is the fact there is no mechanism for dealing with criticality. This can cause problems when tasks with long periods need a high priority. Burns et al [28] provides a real example for a satellite telemetry system, where the software to transfer data to and from the ground station is only needed once a day. The execution of the software within a tight window of opportunity is critical to the system's effectiveness. With the rate monotonic policy the telemetry task would have the lowest possible priority and consequently miss its deadline. To solve the problem using the rate monotonic approach requires the task to be given a much shorter period so that it has a

higher priority, and therefore the task can meet its deadline. However, the wasted resources may cause other tasks to miss their deadline.

The solution to the problem of criticality is the deadline monotonic policy, where task priorities are assigned relative to deadlines. The highest priority is given to the task with the shortest deadline. The original work on the deadline monotonic policy was published by Leung and Whitehead [61]. The deadline monotonic approach is analogous to the rate monotonic approach if all the tasks' deadlines are equal to their period. Leung and Whitehead show the deadline monotonic scheduling approach is optimal for task sets scheduled preemptively when for all tasks, the offsets are zero and their deadline is less than or equal to their period. Using the deadline monotonic approach, the problem of scheduling the telemetry task of the Olympus satellite is solved by giving the task an appropriate deadline. This effectively gives the task a high enough priority to meet its deadline without having to waste resources through unnecessarily increasing the task's period.

There are two restrictions within this computational model that prevent it from being applicable to some systems. The most important is the lack of support for offsets and to a lesser extent the fact that deadlines have to be less than or equal to their period.

### 3.1.6 Timing Analysis for Arbitrary Release Times

In practical systems, it is frequently necessary to offset the execution of tasks from one another. Offsets are used within real systems so that actual requirements and design derived requirements can be met. There are a number of reasons why there is a wish to use offsets in a system, including:

1. Ensuring a particular action instigated by one task has been performed before releasing another task. For example, task A may request data from a hardware device, however, the data may not be available for a time X milli-seconds later. Therefore, an associated task B is required to read the data when it is available, this requires an appropriate separation to be enforced.

2. The spreading of processor resource requirements through a given time frame so that jitter is reduced.

3. Ensuring precedence relations are maintained by not allowing a task to be released until another has completed its execution.

The existing analysis in equations (3.2)-(3.6) may still be applied if the offsets assigned to tasks are ignored. However, the results are pessimistic because the analysis ignores the phasing of task execution.

Some work [54, 62] has already been performed on the timing analysis of task sets featuring offsets. Audsley [54] states the main problem with the analysis of task sets that feature offsets is determining the worst-case release time of each task, i.e. the critical instant. In equations (3.2)-(3.6), the parameter affected by tasks having offsets is the interference. To determine a particular task's critical instant requires its interference to be known. Therefore, the challenge is to derive an approximate measure of worst-case interference that is not computationally complex whilst giving minimal pessimism. Audsley's approach to "inexact" analysis is to provide functions that approximate the interference suffered by a task at any particular point in time. The interference calculated is at least the value obtained through "exact" analysis, which ensures the test is not optimistic.

A problem with Audsley's and Tindell's approach is that the interference equations are complicated to follow and prove, which may make technology transfer difficult. It is felt that part of the reason so much complexity is introduced is because the analysis tries to be truely general purpose. In practice the system being considered may have a relatively uniform nature since its requirements could be a legacy from cyclic scheduling. A challenge is to derive timing analysis that keeps both pessimism and computational complexity low. An investigation of alternative approaches to dealing with offsets is dealt with in Chapter 8. The claim is that a simplified form of analysis can be effective when analysing task sets of the type commonly found in systems.

Related to the need for improved timing analysis is the fact the deadline monotonic priority assignment is no longer optimal when there are offsets. Audsley's thesis [54] provides an optimal approach to priority assignment. The approach involves:

1. performing schedulability analysis,
2. if the analysis shows a task is not schedulable, then swap priority levels with the next highest priority task else finish,
3. if all the possible priority orderings have not been exhausted or stalemate has not been reached, then return to step 1.

Whilst this approach is optimal, it relies on performing many iterations of the timing analysis that is already considered too computationally complex. Therefore, the timing analysis derived in Chapter 8 has to account for the need for a practical priority assignment technique.

### 3.1.7 Task Attribute Assignment

An additional issue borne out of the characteristics of the system considered in section 2.1, is that tasks have to interact in the schedule, e.g. to form transactions. The timing analysis and priority assignment discussed so far only consider individual tasks. This effectively ignores potential interactions that are fundamental to the correct operation of the system. Two different approaches to the problem of uniprocessor task attribute assignment are presented by Gerber et al [63] and Yerrabilli [64]. Task attribute assignment is the process of assigning attributes (i.e. the period, offset, priority or deadline) to tasks so that the timing requirements are met. There is also an approach to task attribute assignment based on priority inheritance.

#### Gerber's Approach

Gerber, Hong and Sabsena [63] have investigated the issue of how to deal with transactions. They show how careful attention to the requirements of intermediate tasks within a transaction may lead to an increased likelihood of the task set being schedulable. Their approach is to use heuristic algorithms to derive

a set of task attributes to meet the system's requirements. The task attributes calculated at this stage may feature non-zero offsets and deadlines not equal to period. A technique, referred to as Fourier elimination, is applied to try to eliminate offsets and make deadlines less than or equal to their period. The aim of using Fourier elimination is to simplify the subsequent implementation and analysis.

The technique whilst powerful, is difficult to understand and justify, therefore a more straightforward techniques would be of benefit. However, the main barrier to the adoption of their work is that the approach assumes all attributes are changeable. This ignores the fact that intermediate tasks may have other functions outside of the transaction. Therefore, changing the period is likely to cause problems.

**Yerrabilli's Approach**

An approach related to that of the previous approach has been developed by Yerrabilli [64]. Again, offsets and deadlines are manipulated using a heuristic algorithm so that transaction requirements are met. However, Yerrabilli makes the distinction that tasks' periods are not free variables. Yerrabilli's approach is different in that it emphasises the need to maximise scalability. In this context, scalability refers to the ability to increase tasks' worst-case execution times whilst still meeting the timing requirements.

The ability to scale the system is an important quality when assigning task attributes. However, the shortcoming of Yerrabilli's technique is that it is difficult to determine how the attributes actually meet the requirements. Analysis can be derived that justifies the requirements are met but the ability to understand how the requirements are met is lost. This ability to understand how the requirements are met is crucial during the certification process when justifying correctness.

**Priority Inheritance Approach**

Control of task precedence is difficult without the flexibility of altering intermediate task attributes. One solution is for the scheduler to control the order in which tasks are executed so that precedence relationships are met. A mechanism would have to ensure that only one of the tasks has permission to execute at a time. A supervisor function, probably within the kernel, would have control of which task. A problem with rate monotonic scheduling theory is ensuring that

higher priority tasks are not delayed more than necessary when they form part of a precedence relationship. Lampson and Redell [65] identified what is known as the priority inversion problem where a higher priority task is blocked by a lower priority task due to the need to conform to a precedence relationship.

The first solution suggested to the problem of priority inversion was the priority inheritance protocol described by Rajkumar, Sha and Lehoczky [66]. The priority inheritance protocol means that a lower priority task can inherit the priority of a higher priority task if it is necessary to prevent the higher priority task being blocked. However, the analysis in section 3.1 caters for cases where blocking may exist. Clearly, when a lower priority task inherits a higher priority, then blocking may be introduced that has to be allowed for in the analysis.

The problem with the priority inheritance protocol is that when a semaphore is used within the software for synchronisation then deadlocks may occur. For example, suppose that at time t1 job J2 locks semaphore S2 and enters its critical section. At time t2, job J2 attempts to make a nested access to lock semaphore S1. However, job J1 a higher priority job, is ready at this time. Job J1 preempts J2 and locks semaphore S1. If job J1 tries to lock semaphore S2, then a deadlock is formed.

A number of solutions have been suggested to the problem of deadlocks with the priority inheritance protocol. The most common solution is based on the priority ceiling protocol by Sha, Rajkumar and Lehoczky [66, 67]. The priority ceiling protocol only allows a task to enter its critical region if it can complete the critical region, i.e. can lock all the necessary semaphores. The priority ceiling protocol allows an executing task that is blocking a higher priority task to inherit the priority of the higher priority task. The ceiling priority is the highest priority task that can lock a particular semaphore.

The problem with the priority ceiling protocol is that a task can be blocked while lower priority tasks complete their execution because the lower priority already have locks on the necessary semaphores. This situation can lead to tasks suffering so much blocking that deadlines are missed. The tasks could suffer a lot of blocking due to the priority being raised in a number of stages as resource contentions are identified. Each time the contention is identified and the priority raised overheads are incurred. Rajkumar, Sha and Lehoczky [68] developed the

semaphore control protocol to help minimise the amount of blocking that can occur within the system while avoiding deadlocks. With this protocol, when a task has locked a semaphore the task priority is immediately raised to the ceiling priority. The ceiling priority is the highest priority of any of the tasks that can lock the semaphore.

The semaphore control protocol reduces blocking by preventing tasks unnecessarily interfering with the task that has locked the semaphore when immediately inheriting the ceiling priority would have prevented it. The timing analysis is simplified since a task may assume only two priorities - its own priority and the ceiling priority. Timing analysis can be performed for the worst case. The worst case is when all tasks other than the one being considered inherit their maximum priority. The drawback of the semaphore control protocol is the potential for significant pessimism within the timing analysis.

There are three problems with the approaches based on ceiling protocols, which are; additional complexity is introduced into the kernel making the design and certification more difficult, the timing analysis is pessimistic and the extra functionality causes overheads at run-time.

Chapter 6 proposes a technique for task attribute assignment that caters for transactions in an effective and understandable manner with the added benefit that the complexity is taken off-line. The technique examines how the correct manipulation of deadlines leads to the task's interaction being correctly controlled.

### 3.1.8 Timing Analysis of Tasks With Arbitrary Deadlines

Other analysis has addressed the need for timing analysis when tasks have arbitrary deadlines [69, 70], i.e. the tasks' deadlines greater than their period. The issue is that the analysis has to be expanded so that multiple instances of the same task can be queued at the same time. The standard schedulability analysis shown in equations (3.2)-(3.6) is expanded as shown in equations (3.7)-(3.8). Equation (3.7) converges when $w_i^{n+1}(q) = w_i^n(q)$. A solution is found when $w_i(q) \leq (q+1)T_i$. This condition relates to a situation where no instance of the

task being considered is waiting to be executed.

$$R_i = max_{q=0,1,2,..}(w_i(q) - qT_i) \tag{3.7}$$

$$w_i^{n+1}(q) = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{T_j} \right\rceil C_j \tag{3.8}$$

where $q$ is the particular instantiation of the task in the range [1,..]
$w_i(q)$ is the worst-case response time of the $q$th instance of task $i$,and
$w_i^n(q)$ is the $n$th iteration of the calculation.

Equations (3.7) and (3.8) iterate for as many instances of a particular task as it takes for latest instance of the task to converge with a worst-case response time less than or equal to the task's period. The analysis is stopped if any task's response time exceeds its deadline. When considering a particular instance, the response time is calculated iteratively. If the response time is greater than or equal to the task's period, then another instance of the task is released. However, the earlier instance of the task is always executed first. The latest instance of a task completing before another instance is released means that no instances of the task remain on the run queue. The actual worst-case response time is the maximum response time of all the task's instances before convergence is achieved.

### 3.1.9 Analysing Kernel Overheads

The analysis presented so far has assumed an ideal system where there are no kernel overheads associated with task release. This section is to survey how the kernel overheads can be analysed drawing on two principal references [52, 71].

Irrespective of the task release mechanism employed, there is a cost associated with the context switches into and out of a task. Independent of how many times a task is preempted, the task only suffers two context switches - the one into the task and the one out of the task. The reason is the context switches associated with each preemption are analysed as part of the task causing the preemption and not the task suffering the preemption. Equation (3.9) presents analysis for the context switches. The analysis basically increases each task's worst-case execution time to allow for context switches. Equation (3.10) is a

simplified version that assumes the cost of a context switch is the same into and out for all tasks.

$$C_i' := C_i + C_{s\_in} + C_{s\_out} \tag{3.9}$$

where $C_i$ is the actual worst-case execution time of task $i$,
$C_i'$ is the modified worst-case execution time of task $i$ allowing for the context switches,
$C_{s\_in}$ is the worst-case execution time of the context switch into the task $i$, and
$C_{s\_out}$ is the worst-case execution time of the context switch out of the task $i$.

$$C_i' := C_i + 2C_s \tag{3.10}$$

where $C_s$ is the worst-case execution time of a context switch into or out of any task.

A commonly adopted way of releasing tasks is based on a regular clock tick. When the clock tick arrives, tasks that are waiting to be released are released and the highest priority task is executed. If the tasks are released using a clock tick, then there are many ways in which the cost of the clock tick can be modelled. However, there are two basic variants for the modelling; as a single task or as multiple tasks.

1. *Single Task*

   A single task can be used to model the clock tick. The single task would have a period the same as the clock tick and a worst-case execution time that encapsulates the time taken to update the run queues. The task used to update the run queue would be the highest priority in the system, and could not be preempted so it suffers no blocking.

   The worst-case time to update the run queues is when all the tasks are released at the same time. The principal problem with the single task approach is that it is clearly pessimistic because tasks are not always released as often as the clock tick. However, the results can be improved using

equation (3.11) (taken from [71]) to provide a less pessimistic estimate for the worst case time to update the run queues. Equation (3.11) allows for the relationship between the tick period and the minimum period of the set of tasks.

$$C_{clock} = MC_{first} + (N - M)C_{sub} \tag{3.11}$$

where $N$ represents the number of tasks in the system,
$T_{min}$ is the shortest period of all tasks,
$T_{clk}$ is the period of the clock tick,
$C_{first}$ is the cost of releasing the first task,
$C_{sub}$ is the cost of releasing the subsequent tasks, and
$M = \left\lceil \frac{T_{min}}{T_{clk}} \right\rceil$

2. *Multiple Tasks*
   For every task in the system, an additional task is used to model the updating of the run queue. Each additional task has the same iteration rate as the actual task, and a worst-case execution time equivalent to the worst-case overheads (obtained via analysis) for the release of the particular task.

Tasks can also be released based on the value of a real-time clock, i.e. in a time driven manner. At pre-defined points in a task's execution, the task would stop executing while the run queue is updated and the highest priority task is executed.

The time driven model can be analysed by each task having its worst-case execution time augmented by an amount $C_{COOPi}$, where $C_{COOPi}$ is the worst-case time to determine a task is runnable and then release it. In practice, $C_{COOPi}$ is not a constant. Typically task information is stored in priority order and the tasks are searched to determine the highest priority runnable task. Therefore, $C_{COOPi}$ could be expressed as shown in equation (3.12). Frequently the implementation is optimised to cut down the search time, leading to the execution time in equation (3.12) being pessimistic. For example, if no tasks are released while the current highest priority runnable task is executed, then the search can be optimised. Instead of starting with the overall highest priority task, the search

can start with the next highest priority task after the task that has just finished executing. However, equation (3.12) still represents the worst case.

$$C_{COOPi} = p_{i-1}C_{SEARCH} + C_{RELEASE} \tag{3.12}$$

where $C_{RELEASE}$ is the worst-case time to release the next task,

$C_{SEARCH}$ is the worst-case time to determine a task isn't runnable, and

$p_i$ is the priority of task $i$, the range is [1,N] with 1 being the highest.

For reasons of simplicity, $C_{COOPi}$ is often taken as a constant $C_{COOP}$ to make the analysis less complicated. The value chosen for $C_{COOP}$ would have to be the value of $C_{COOPi}$ for the lowest priority task since it has the largest value. The resultant worst-case execution task ($C_i'$) for a task released using the time driven approach is given in equation (3.13).

$$C_i' := C_i + C_{COOP} \tag{3.13}$$

### 3.1.10 Case Study - Olympus Attitude and Orbital Control System

There have been a number of attempts to put fixed priority scheduling into practice on real industrial systems. This section looks at one case study, which is the Olympus Attitude and Orbital Control System (AOCS) [28, 72]. The Olympus satellite was launched in July 1989 as the world's largest and most powerful civil three-axis-stabilised communications satellite. The AOCS sub-system exists to acquire and maintain the desired spacecraft position and orientation.

A case study was commissioned to investigate the technical issues and benefits of using fixed priority scheduling on a ground based demonstrator. The case study is based on the normal mode of the AOCS system, which is the most complex mode and is used for the greatest percentage of the satellite's lifetime. The application selected contains many typical features of real-time software, i.e.

1. Periodic tasks
2. Sporadic tasks
3. Hard real-time tasks

4. Soft real-time tasks

5. Background tasks

6. Communications over a databus

One feature of real-time software that is not part of the case study is transactions, which is considered to be a significant exclusion. Comparative testing with the original Olympus satellite equipment yielded the following conclusions:

1. there was no significant difference in the quality of control, and

2. the systems were judged to be equivalent in their signal responses and noise magnitudes, except for one functional area.

The study did highlight some disadvantages of the technique that can lead to increased cost. These are:

1. overheads are increased by the run-time support system to support Ada tasking, and

2. there is a need to invest in support tools and training to achieve the maximum benefits from the approach.

In addition, the papers on the case study did not consider the issues of reuse and certification when proposing a solution. These issues are considered fundamental if the solution is to be accepted. The case study suggests the fact that the positive benefits far outweigh the disadvantages. These include:

1. a sound engineering approach with a mathematical basis that provides more exact results with reduced effort,

2. flexible run-time scheduling allows changes to be made in the application structure without the costly re-development of the cyclic scheduler, and

3. tool support enables the techniques to be used by engineers rather than academic theoreticians.

## 3.2 Fixed Priority Scheduling of Distributed Systems

The purpose of this section is to survey the existing literature available on fixed priority scheduling and timing analysis for distributed systems. Techniques utilising the existing uniprocessor timing analysis are of particular interest. The reasons are:

1. There may be a great deal of time, money and effort invested in the existing theory and tools. It would be beneficial to allow the tools to be reused.

2. Treating each processor individually eases the synthesis and maintenance effort by facilitating a modular approach to design.

### 3.2.1 Release Jitter Based Approach to Distributed Scheduling

The early work on an event-driven model for scheduling of distributed systems was performed by Tindell and Clark [69]. Their approach uses periodic tasks for the first task in the transaction. Subsequent tasks or messages are triggered as sporadic tasks when the preceding task has been completed. For the purpose of the timing analysis, sporadic tasks are modelled as periodic tasks, with period equal to their minimum inter-arrival time. To account for the variability of sporadic task's release, each task is given a release jitter equal to the worst-case response time of the preceding task. (It should be noted that the case where two or more consecutive tasks of a transaction execute on the same processor is different. In this case the tasks would be released simultaneously and priorities used to control the ordering of task execution, rather than releasing a later task after an earlier task has completed executing.)

Using the release jitter based approach, the system timing properties may be accounted for at the uniprocessor level. The subsequent tasks/messages are

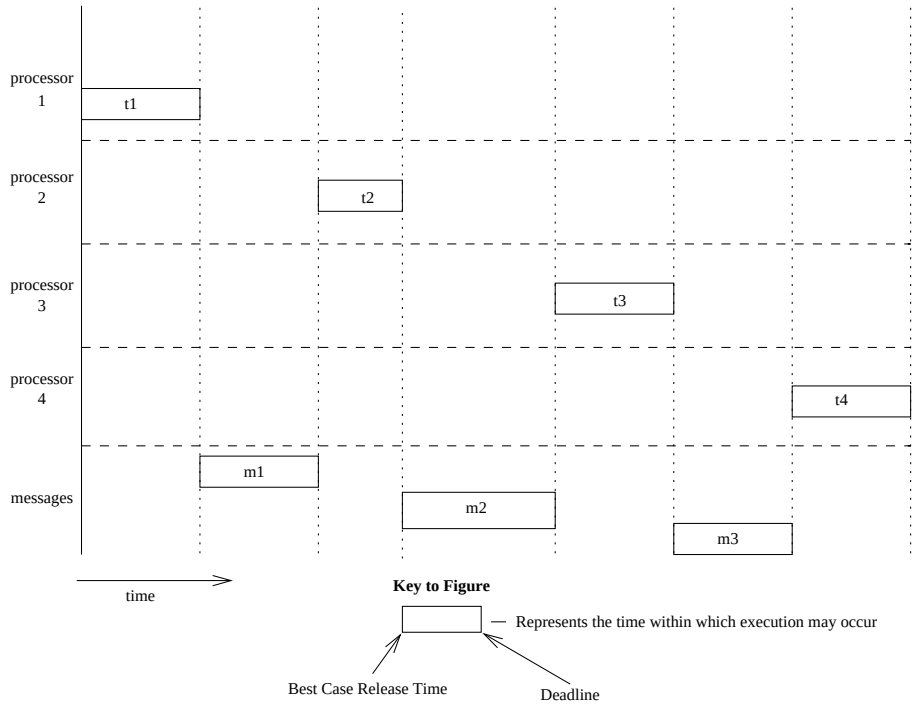


Figure 3.1: Diagram to Illustrate the Timing Analysis of a Transaction

modelled as periodic tasks with characteristics:

$$\textbf{period} = \textbf{transaction period} \tag{3.14}$$

$$\textbf{release jitter} = \textbf{worst-case response time of the previous task or message in the transaction} \tag{3.15}$$

The relationship between release jitter and completion time is illustrated in Figure 3.1. Figure 3.1 shows how for a particular transaction, messages are released by tasks and tasks are released by messages. For example, the task *t3* is released after the worst case arrival time of message *m2*. Therefore, the release jitter of task *t3* is equal to the worst-case arrival time of message *m2*. Equation (3.16) helps to show how the release jitter accounts for distributed transactions.

$$J_{t+1} = R_t \tag{3.16}$$

where $t$ refers to the $t$th task/message in the transaction, and

$R_t$ is the worst-case response time of task $t$ from the start of the transaction.

Equations (3.5) and (3.6) provide analysis that caters for release jitter. The transaction's end-to-end deadline requirement is verified by considering the worst-case response time of the last task in the transaction. If the worst-case response time is less than or equal to the end-to-end deadline of the transaction then the requirement is met. Therefore, the uniprocessor schedulability analysis in section 3.1 can be used to verify the system's timing requirements.

Tindell and Clark [69] show how the timing analysis for arbitrary deadlines given in equations (3.5) and (3.6) may be used in cases where the transaction's deadline is greater than the tasks' (that form the transaction) periods. Their approach modifies the tasks' (that form the transaction) deadlines to be equal to the transaction deadline. Then, verifying the tasks' deadlines, using equations (3.5) and (3.6), as met also verifies the transaction's deadline.

Harbour, Garcia and Guiterrez [70] demonstrate that the approach is pessimistic. If a task's release jitter is greater than its period, then multiple task instances are assumed and the arbitrary deadline timing analysis presented in section 3.1.8 is used. In practice, multiple task instances do not exist since the tasks have deadlines less than or equal to their period. Harbour's approach is to maintain the original task deadlines. Then, the verification of transactions is dealt with separately to the schedulability analysis of tasks.

Harbour et al's approach requires the relevant instance of a task, with respect to the transaction, to be determined so that the response time of the transaction can be calculated. The relevant instance is chosen such that its release time is greater than or equal to the worst-case response time of the preceding task in the transaction. The advantage of Harbour et al's approach is the analysis is less pessimistic. The disadvantage is that extra analysis is required for verifying transaction requirements are met. The verification of transactions can be automated and the analysis is understandable. Therefore, Harbour et al's approach is a viable technique.

The advantage of the release jitter approach is the existing uniprocessor analysis may be used. However, there are four principal disadvantages of the release jitter approach, which are:

1. The use of sporadic tasks in safety critical systems is frowned upon due to the inherent difficulty in analysing their operation, i.e. the effect of task release at arbitrary points in time, and the effect of omission and commission failures in the release mechanism.

2. The use of sporadic tasks present an additional problem when implementing distributed transactions. The computational model means the tasks later in the transaction's order have more release jitter. However, the output jitter of a transaction is frequently important. The output jitter is equal to the variation in the time of the output from the transaction. This is often taken to be the variation in the completion time of the last task in the transaction. The variation in the last task's completion time is greater than or equal to the last task's release jitter. Therefore, the approach described in this section is considered prohibitive for many applications.

3. The pessimism associated with this approach is considered prohibitive. The release jitter increases with the number of tasks in the transaction, and reaches a stage where the system is likely to be unschedulable.

4. The release jitter approach is not robust to change. Small changes on one processor can easily lead to the system requiring re-verification.

### 3.2.2 Offset Based Approach to Distributed Scheduling

An alternative approach to distributed timing analysis is based on the exact analysis approach proposed by Leung and Whitehead [61], which is discussed in section 3.1.2. Within the context of distributed systems, the exact analysis was first presented as the Phase Modification Protocol by Sun and Liu [73]. The approach serves as an ideal way of assessing the effectiveness of any approach developed. The exact form of analysis takes the approach of showing every individual release of each task on all the processors and all the messages on the databus are schedulable.

The Phase Modification Protocol approach provides both implementation and verification advantages over the release jitter approach. The Phase Modification Protocol approach uses only periodic tasks within the implementation. The precedence constraints of the transactions are enforced using offsets by giving the task or message an offset equivalent to the worst-case response time of its predecessor.

The drawback of an offset based approach is that the implementation requires the use of a global time base. An additional benefit of the approach is the correct phasing of tasks removes the need for protocols, such as the priority ceiling protocol. An appropriate phasing of tasks prevents two or more tasks trying to access the same resource at the same time. Therefore, resource contention is avoided and hence priority inheritance is not needed. The obvious disadvantage is the maintenance of an appropriate phasing is difficult for the same reason as with cyclic scheduling, discussed in section 2.5.2.

Figure 3.1 can be again used to show how the time during which a task or message may execute is controlled. For example, in the case of task *t2* the duration of allowed execution commences when message *m1* has definitely arrived, i.e. $O_{t2} = R_{m1}$. Message *m2* is then scheduled for when task *t2* has completed, i.e. $O_{m2} = R_{t2}$. By giving a task an offset such that its dispatch (or release for simplicity) is always greater than the worst-case response time of the event trigger (in the case of task *t2* the worst-case arrival time of the message *m1*) then precedence is maintained, even across a distributed system. Equation (3.17) shows how the Phase Modification Protocol accounts for the distributed transactions.

$$O_{t+1} = R_t \tag{3.17}$$

where $O_{t+1}$ is the offset of task $t + 1$ from the critical instant

The Phase Modification Protocol adopts this approach so the release jitter caused by variable release times of sporadic tasks is removed. There are two drawbacks associated with the Phase Modification Protocol. Firstly, new analysis is required, which may be difficult to understand and may have high computational complexity. For realistic systems, the analysis can become intractable due to the least common multiple of the periods being very large, particularly when the tasks have periods that are co-primes. Secondly, the design and verification of the system's timing aspects are not robust to change, i.e. a change to one

task's worst-case response time leads to system wide change. A simple change of functionality related to one processor may lead to changes in task attributes throughout the system, as well as a complete re-analysis.

### 3.2.3 Task Attribute Assignment for Distributed Systems

A number of approaches have been proposed to the problem of task attribute assignment for distributed systems. The aim of these techniques is to assign task attributes based on heuristic methods rather than the results of timing analysis. The problem of basing the task attributes on the results of timing analysis is a lack of robustness to change.

One of the first papers that deals with the issue of task attribute assignment for distributed systems was produced by Bettati and Liu [74]. Their paper deals with the scheduling of distributed systems using flow shop scheduling. Flow shop scheduling is where tasks are executed on a "first come first served" basis. Bettati demonstrates how assigning local deadlines evenly across all tasks in the transaction is a practical technique for meeting the timing requirements whilst minimising the effect of change. The approach states that if a transaction deadline is $X$, then each task is assigned a non-overlapping window of $X/N$ (where N is the number of tasks in the transaction). Therefore, the third task in the transaction would have an offset of $2X/N$ and a deadline of $3X/N$. The approach was based on a simplifying assumption that all tasks have identical worst-case execution times. If the assumption is true, then the approach is optimal. Clearly, the assumption is not realistic.

Following on from Bettati's work, Natale and Stankovic [75] produced an approach called "time-slicing", which is intended for fixed priority scheduling and does not make the simplifying assumption. Instead of uniformly sharing the available time across the tasks of the transaction, the time-slicing approach shares the time so that each task has the same laxity. In this case, laxity is the difference between a task's execution time and its deadline. The aim of the technique is to increase the chance of schedulability and increase the resilience to change. Natale and Stankovic shows the technique to be optimal in the sense that the laxity of each task is maximised.

Shin [76] demonstrates how time-slicing can be more effective if the size of execution time slices are chosen in an adaptive manner. Shin uses a heuristic approach based on tasks' computation time. The approach leads to an increase in the likelihood of schedulability with an acceptable increase in computational complexity.

An alternative approach is proposed by Harbour and Garcia [77]. This is based on a number of iterations of task attribute assignment and schedulability analysis in order to derive a set of task attributes. Initially, task attributes are assigned at a local processor level. Schedulability analysis is performed to determine whether the requirements are met. Then, each task has its attributes changed using a heuristic that is based on by how much a deadline is missed. The greater the amount of time by which a deadline is missed, the more a task's deadline is increased before the task set is re-analysed. Harbour's approach provides a simple approach to choosing task attributes but has the drawback the solution can take a long time to compute.

A great deal of work has been performed on search algorithms, such as simulated annealing and genetic algorithms [78, 79]. These approaches can provide solutions to the scheduling problem. The drawbacks of using search algorithms are that the searches can take a long time to complete, and the results are hard to justify and explain because there is no clear reason for the results. The problem with all the techniques is that when the system is changed a complete re-synthesis is necessary. This necessitates a complete re-verification, which makes the cost of regression testing too great for practical purposes.

Chapter 9 examines the problems of distributed timing analysis in greater depth so that improved timing analysis can be derived. Significant amounts of simulation are used for this purpose. The approach derived is based on the use of offsets to control precedence. The approach makes use of the timing analysis in Chapter 8.

## 3.3 The Time Triggered Architecture

This section discusses the work performed by the University of Vienna on the Time Triggered Architecture. The purpose is to build an understanding of how other work has addressed the needs of safety critical systems.

The MARS kernel is often considered the most appropriate published approach for safety critical systems. The reason the MARS project is viewed as superior to many of the other projects is that it is the only work specifically targeted at the safety critical systems domain. Other approaches make compromises between dependability, analysability and performance to obtain a general-purpose architecture. The MARS work has also paid a great deal of attention to providing a complete picture, including; fault tolerance, dependability, timing analysis and tool support. These issues are important to safety critical systems. To date, the MARS work is still considered the most relevant to this thesis.

The Time Triggered Architecture (TTA) has evolved from the MARS kernel project taking advantage of the lessons learnt. Therefore, this section briefly reviews the work of both of these projects. The TTA project is intended as a complete process and system solution to the problems of designing embedded distributed systems for the safety critical systems market. The best source of information on the TTA work is a book by Kopetz [80].

The MARS kernel has been developed for more than 10 years with the intention of developing an infrastructure to meet the system's requirements (including certification) in a methodical tool assisted manner. The basic architecture for TTA is the system is split into a number of clusters. Each cluster has autonomous responsibility for a set of system requirements. Within a particular cluster, the system has hard real-time requirements. However, data flow between the clusters does not have to be hard real-time. Therefore, our principal interest is the scheduling and timing analysis within a cluster.

Task scheduling is performed using the cyclic scheduling technique, and message scheduling using the Time Division Multiple Access (TDMA) technique. TDMA is where a sequence of slots are executed repeatedly. Each slot is of known size and occurs at a pre-defined time. A single message is assigned to each slot. Therefore, cyclic scheduling and TDMA can be considered as similar. To prevent the criticism related to synthesis and maintenance of cyclic scheduling discussed

in section 2.5, tool support [81] is available. Unfortunately, the tool does not allow for the need to minimise the amount of regression testing during change control, which is a problem when using tool support for schedule synthesis.

A major part of the MARS design ethos is ensuring that the kernel and system operates reliably. At a system level, all the components have at least two redundant components. The processing nodes handle their redundancy by two nodes transmitting their outputs and one node remaining silent. When the silent node detects an error in one of the two other nodes it stops being silent. The errant node is then silenced to provide error containment by a hardware inhibit of its connection to the communications bus. The technique relies on a trusted voter to identify faults. The voter is the means of comparing similar values produced by different sources to determine if an error has occurred.

To ensure that the communications data is sufficiently reliable, data is communicated twice across the network. The reason is that the probability of two data items being corrupted is considerably less than the probability of a communications node failing. Therefore, when performing reliability calculations the effect of node failure can be ignored and the calculations can concentrate on data corruption for which analysis already exists. In addition, the MARS kernel adopts a communications protocol, referred to as a membership service [82], for determining whether any processing nodes have failed. In any distributed system, the membership set may change over time because an active node departs from the membership set (e.g. node failure) or an inactive node may join (e.g. a node that previously failed may have recovered or has been repaired). The aim of the protocol is to guarantee that within a time interval every active node has a consistent knowledge about the membership set.

The viewpoint that the MARS and TTA work is the most relevant to safety critical system is based on the completeness of the solution. Unlike other work, detailed design of important dependability issues such as a fault tolerant clock synchronisation, timely membership service, a reconfiguration management, and provision of fail silence in the temporal domain, has been performed. It is the completeness that raises the confidence in the approach taken. The principal problem with the Time Triggered Architecture work is that it is based on a cyclic scheduler.

## 3.4 Integrated Modular Avionics

The Integrated Modular Avionics (IMA) initiative is an attempt to solve many of the problems currently faced by avionic systems [8, 83]. The problems include; obsolescence caused by components going out of production during the lifetime of the system, and the large size of systems necessitating the design being split into manageable independent parts. The IMA system architectures being considered provide modularity and technology transparency in order to solve the problems.

Modularity is where the architecture provides facilities to control how different partitions of software communicate through defined interfaces. The main aims of modularity are; to allow different parts of the software to be written in isolation but integrated with ease, to ease the problems of change by reducing the scope of change, and to allow different integrity levels of software to exist safely on the same processor. The objective of technology transparency is to allow the processor the software is executing on to be changed with the minimal of rework. To achieve technology transparency, a portable code is required, such as provided by Java byte-code [84], that can execute in a similar manner on a wide range of platforms. The aim of technology transparency is to prevent obsolescence.

Fletcher et al [85] recognises how IMA provides a significant number of technical and certification challenges. A key challenge is the scheduling and timing analysis has to be abstract in nature so that the maximum reuse of the system and certification evidence occurs when the underlying platform is changed. Grigg and Audsley [86] has proposed a solution based on reservation-based scheduling [87] that may provide a solution. However in the short term, fixed priority scheduling is seen as the most likely candidate [85]. There is little likelihood of cyclic scheduling supporting technology transparency because it has low robustness to change [86].

## 3.5 Summary

This chapter has surveyed the existing work on fixed priority scheduling for both uniprocessor and distributed systems. The chapter has also looked at the Time Triggered Architecture work, which is currently considered to be the "state of the art" for safety critical systems. The survey has also investigated how future influences, such as IMA, may influence the scheduling approaches derived.

The survey highlighted a number of problems with the current approaches that necessitate a change from cyclic scheduling and inhibit the use of the traditional form of fixed priority scheduling. The chapters so far give an indication that fixed priority scheduling, with some modifications, is suitable for use in the safety critical systems domain. The remainder of the thesis is to assess how fixed priority scheduling may be used, and justify the claim that it is an appropriate technique.

# Chapter 4

# The Transition from Cyclic Executive Scheduling to Fixed Priority Scheduling

There are four basic aims of this chapter:

1. Section 4.1 investigates whether fixed priority scheduling represents a solution to the problems of cyclic scheduling that are highlighted in section 2.5.

2. Section 4.2 considers the problems of fixed priority scheduling, for example those found in the Olympus AOCS case study of section 3.1.10 and in Locke's paper [6], and whether they can be solved.

3. Section 4.3 examines how the fixed priority scheduling technique may be used to replace the cyclic scheduler in an existing system. The implications of any changes to the "standard" fixed priority scheduling model are considered.

4. Finally, section 4.4 highlights the remaining work that is required for fixed priority scheduling to form a complete solution.

## 4.1 Is the Fixed Priority Scheduling the Solution to the Problems of Cyclic Scheduling?

In section 2.5 a number of problems with cyclic scheduling are raised: maintainability, efficient use of resources, and supporting future requirements. The problems of cyclic scheduling are derived from issues raised in a paper by Locke [6] and through contact with industry. Locke's paper describes the weaknesses of cyclic scheduling and identifies fixed priority scheduling technique as the smallest technological leap capable of solving the problems of cyclic scheduling. Locke's view that fixed priority scheduling is the "natural" progression from cyclic scheduling is a commonly supported belief. Two supporting references for Locke's viewpoint are Stankovic [88], and Burns, Audsley and Tindell [89], however there are many more. The claim is that fixed priority scheduling can solve the problems of cyclic scheduling. To justify the claim, the three problems of cyclic scheduling are considered in turn to examine how fixed priority scheduling may provide the desired solution.

### 4.1.1 Efficient Use of Resources

Section 2.5.1 describes the three basic reasons (limited iteration rates, lack of sporadic tasks, and the waste of resource at the end of a cycle) why the cyclic scheduler is considered to use resources inefficiently. All the reasons are related to the restricted computational model. The fixed priority scheduler eradicates these problems because the computational model is not limited by the cyclic arrangement and the non-static schedule. Also, the fact the schedule is not arranged using a cyclic structure removes the need to reserve time at the end of each cycle, which is wasted resource. The only time the processing resource is unused in fixed priority scheduling is when the run queue is empty.

The lack of a cyclic structure and non-static ordering means that periodic tasks can have any iteration rate. The non-static ordering also allows sporadic tasks to be implemented. The increased flexibility of the computational model means the actual requirements may be implemented, rather than those imposed by the scheduler. This leads to a more efficient use of resources and also the possibility of a system that performs better.

### 4.1.2 Maintainability

Section 2.5.2 highlights how the maintenance problem of the cyclic scheduler arises from the need to support task and transaction requirements, which leads to difficult synthesis and change control problem. There are four principal reasons why fixed priority scheduling is easier to maintain than cyclic scheduling; ease of synthesis, robustness to change, ease of verification, and reduced regression testing.

#### Ease of Synthesis

A fixed priority scheduler is easier to synthesise than the cyclic schedule. Garey and Johnson [45] show the bin-packing problem of the cyclic executive scheduler to be NP-complete. Whereas, Leung and Whitehead [61] shows the problem of synthesising fixed priority schedulers is pseudo-polynominally complex. A fixed priority scheduler is easier to synthesise than a cyclic scheduler because synthesising priorities is simply a matter of assigning a place in an order. However, with a cyclic scheduler each task is assigned a number of places (dependent on the task's iteration rate and the minor cycle rate) in a number of minor cycles (dependent on the minor and major cycle rates) as well as having to order tasks in each individual minor cycle.

#### Robustness to Change

The fixed priority scheduler is more robust to change than the cyclic scheduler. The reason is that with a fixed priority scheduler all that matters is whether the timing requirements are met. A change to the scheduler is only necessary if the requirements are no longer met. Whereas with the cyclic scheduler the timing requirements being met is not enough. A change to a task's worst-case execution time may lead to a minor cycle overflowing that also necessitates the schedule to be changed. The extra condition to be upheld means that more changes to the schedule occurs, particularly in a heavily loaded system. In addition, with a fixed priority scheduler a failure to meet a deadline could be solved by simply swapping two adjacent priorities. Whereas, a cyclic schedule has to be completely re-synthesised. Therefore, the fixed priority scheduler is easier to change than the cyclic scheduler.

#### Ease of Verification

The verification of a fixed priority scheduler tends to take less effort than a cyclic scheduler. The reason is a fixed priority scheduler is verified using analysis, whereas a cyclic scheduler is often verified through test. Test-based verification is why a static run-time ordering is important because the behaviour is deterministic. Analysis can be performed automatically in a short amount of time with a minimum amount of effort. Test is considered to be a much more expensive activity than analysis.

#### Reduced Regression Testing

Related to the previous point, with a fixed priority scheduler the verification of the system is not dependent on a static run-time ordering. Therefore when a fixed priority scheduler changes, there is no need to repeat the functional testing of the system. However when a cyclic scheduler is changed, all the system's data-flow is changed. Hence, the effort required to perform regression testing of the functional requirements could be considerably increased.

### 4.1.3 Supporting Future Requirements of the System

Section 2.5.3 describes how the cyclic scheduler is likely to prohibit the future systems that need to be developed. Consideration of IMA systems in section 3.4 and in reference [86] provides an example of where a cyclic scheduler is inappropriate. The fact the fixed priority scheduler's computation model is less restricted than the cyclic scheduler makes it more appropriate for the types of system that are likely to be needed.

An important part of this claim is related to the fact that systems are likely to become more de-centralised in nature. De-centralising the system makes maintenance significantly more complex, eventually leading to the cyclic scheduler becoming practically infeasible. The problem of maintaining a set of minor cycles on a single processor to meet task and transaction requirements is difficult enough. However, if there are many processors with a communications bus connecting them, then the complexity becomes even greater. The reason is that all the schedules, including the one for the databus, must interact together to meet the distributed transactions. A schedule change on one processor could cause all

the system's schedules to need re-synthesis. The system being scheduled could have to support many transactions. A move to IMA type systems involving multi-company projects means it is important to actually partition the systems engineering problem, which includes the scheduler.

Another reason the fixed priority scheduler is more suitable for future systems is the computational model only has restrictions by design, this means the system is better placed to support future needs. Any restrictions would probably arise from the difficulty in certifying the general computational model. The restrictions would be traded-off against the needs of the system.

## 4.2 Solving the Problems of Fixed Priority Scheduling

*If fixed priority scheduling is capable of removing the problems of cyclic scheduling, why has it not achieved widespread adoption?*

The reason could be the fact cyclic scheduling has some advantages over fixed priority scheduling. Locke [6] also highlights some disadvantages of fixed priority scheduling, which are: it is frequently reliant on the Ada tasking model whose characteristics are difficult to predict, and the kernel overheads tend to be larger. However, the main disadvantage of fixed priority scheduling is that the safety critical systems domain is always sceptical and resistant to any change from current practice.

### 4.2.1 Tasking Model

Locke [6] states the implementation of the Ada 83 tasking model [90] is hard to predict. The reason was originally highlighted by Sha and Goodenough [47] who state that even though the Ada tasking model requires priorities to be specified, the entry rendezvous queue is exercised in First-In First-Out (FIFO) order. A rendezvous is a means of achieving synchronous communication by the receiver acknowledging that it is ready to receive a message and that it is the correct recipient. Whilst it can be argued any scheduling model is predictable, after all

any computer firmware is based on logic, FIFO makes it harder to analyse and synthesise the schedule than even cyclic scheduling.

The disadvantage of the tasking model being difficult to predict can simply be overcome by the engineers implementing their own scheduler. However, Locke's [6] observations were made with respect to the Ada 83 version [90] of the Ada language. The difficulties associated with the tasking model in Ada 83 may be avoided by using the Ada 95 version of the language standard [91] that includes a revised tasking model as well as support for important scheduling paradigms. These include priority inheritance implemented with priority ceiling protocols. Despite the change, the SPARK subset of Ada 95 [92] still precludes the use of tasking models. An equally strong influence is the desire to make the minimum changes deemed necessary from the SPARK version of Ada 83. However, an alternative safe subset of Ada 95 [93] proposed by the ISO WG 9's HRG committee includes the use of a subset tasking.

Despite the updated tasking model of Ada 95, the use of bespoke schedulers is again preferred. The main reason is the tasking model may not be used is that the implementation may be over complex for a particular application and it is part of the compiler. Therefore, there would effectively be a COTS software product in the system. There are a number of reasons why a Commercial Off-The-Shelf (COTS) product is often not considered appropriate for safety critical systems. The main reasons is that the product is rarely designed for the domain so it cannot be trusted and it would not be cost effective to re-engineer the product to the appropriate standard. Also, the product would carry a lot of baggage (i.e. functionality) that is not needed. This increases kernel overheads and the certification effort [94]. The latter point would make the criticism of higher kernel overheads with fixed priority scheduling [6] more apparent.

For the reasons discussed in this section, this work is to assume a bespoke scheduler is produced. Part of the reason for this decision is that cyclic schedulers are normally implemented in this manner. Therefore, the likelihood of changes being necessary to the software of the tasks is reduced. In addition, the work can then investigate how best to support the tasking model from the perspective of the infrastructure.

### 4.2.2 Kernel Overheads

The overheads incurred with a fixed priority scheduler rather than a cyclic scheduler are generally increased. The reason is that all a cyclic scheduler has to do is execute tasks in a defined order. However, a fixed priority scheduler has a great deal more functionality, including releasing tasks after determining the tasks are runnable, and searching for the highest priority runnable task that is then dispatched.

To alleviate the problems of kernel overheads, there are two issues to be addressed. The first issue is how the kernel should be designed bearing in mind the four criteria (in particular the reuse criterion) for successful technology transfer defined in section 1.3 as well as the need to minimise overheads. The reuse criterion is important because the existing hardware and software (including applications) should be used within the new infrastructure with a minimal amount of rework. However, the ability to implement the system's timing requirements should not be forgotten. The second issue is how pessimism in the analysis of overheads can be reduced. A fundamental part of the transition from cyclic scheduler to fixed priority scheduler is the development of a kernel that can be certified. There are two main design aims for the kernel; the ability to reuse the existing hardware architecture, and to allow the system's timing requirements to be met by having reasonable overheads and by having a responsive infrastructure. The overheads should be appropriate so that a powerful scheduling policy is not handicapped by the kernel overheads being so large that the tasks become unschedulable. A responsive infrastructure is one that releases tasks in sufficient time for their deadlines to be met.

The majority of work on fixed priority scheduling assumes an ideal computational model with no overheads and no failures. Therefore, there is a need to investigate how best to implement and analyse the kernel. Chapter 5 is to investigate the issues related to the kernel.

### 4.2.3 Predictability versus Determinism

One of the key differences between a cyclic scheduler and a fixed priority scheduler is that a cyclic scheduler is deterministic, whereas a fixed priority scheduler is

predictable. The difference is a cyclic scheduler has a static task ordering and a fixed priority scheduler has a dynamic task ordering. This causes significant impact in the area of verification. As a consequence, a fixed priority scheduler has to rely on analysis whereas a cyclic scheduler can rely on either test or analysis.

A positive aspect of this change is that analysis actually provides more accurate (i.e. better) results. However there are two problems, which are; any change of the type of certification evidence provided is a major step that needs justifying, and the change also increases the number of paths when performing functional testing. Any approach derived for the fixed priority scheduler has to address these problems and significant attention needs to be paid to justifying that the change is safe.

## 4.3 How Can Fixed Priority Scheduling Support the Domain's and Application's Requirements?

The purpose of this section is to examine how fixed priority scheduling could be used in a system that currently uses a cyclic scheduler. There are five parts to the discussion, which are:

1. establishing a task execution model,
2. determining whether Deadline Monotonic Priority Ordering (DMPO) is optimal for the execution model,
3. establishing an appropriate task release model,
4. tailoring the timing analysis for the overall computational model, and
5. determining whether DMPO is optimal for the overall computational model.

These are dealt with in the following sub-sections.

### 4.3.1 The Choice of Execution Model

Most scheduling texts generally assume the fixed priority scheduling policy is implemented using a preemptive flow of control, and support is provided for both periodic and sporadic tasks. Timing analysis is widely available, in many forms, for the general model. Chapter 3 provides a survey of fixed priority scheduling, including how it may be analysed. However, in many ways the verification of the timing characteristics is a relatively small job compared to that of verifying the functional properties. Therefore, the wider implications of the actual scheduling model are considered in this section.

The computational model of the cyclic scheduler is based on a non-preemptive flow of control. Non-preemptive scheduling is where a task is always executed to completion. The principal difference between preemptive and non-preemptive scheduling is a task's execution can be preempted at any time by the release of a higher priority task. This leads to a greatly increased number of possible program paths, which means functional testing is more difficult to fully achieve. More importantly, data flows and updates could be interrupted causing a task to be preempted when a data calculation is only partly finished. If data that is in a transient state is used, then the effect could be difficult to determine.

The transient state issue could be solved under a preemptive scheme by defining critical sections of code and preventing preemption when execution enters the critical sections. However, the application and the scheduler software would become more complex. To make the transition from a non-preemptive to a preemptive system requires the existing software design and implementation to be thoroughly examined to determine where problems could occur. Some or all of the software would have to be modified or re-implemented to solve the problems. An additional advantage of having a non-preemptive scheduler is that there is no longer a need to preserve the context for a task when its execution is suspended or finishes.

The real benefit of preemptive scheduling can only be obtained through a completely new design because of the chance given to re-allocate functionality to tasks. For instance, section 2.5 highlights how the non-preemptive execution of software in a cyclic scheduling framework leads to software being broken up into small parts. By re-allocating functionality a more natural decomposition of

functionality to code can be obtained, leading to improved maintainability. A potential problem of taking this strategy is that the reuse criterion for successful technology transfer, introduced in section 1.3, is broken.

An additional barrier to be overcome is the culture of the industry, which generally uses test-based verification. Despite the fact that fixed priority scheduling provides analysis that is quicker and more effective, there is still a desire to test systems. A particular concern is the need to test the system from a functional perspective. Part of the reason a cyclic scheduler is currently used is the static ordering of tasks means the system is deterministic. With a preemptive scheduler, there is clearly a significant increase in the potential number of program paths making complete path coverage less likely. Making the testing problem more difficult further reduces the likelihood of successful technology transfer. Section 2.2 also discusses how the certification authorities are reluctant to allow any interrupts because of the increased certification effort that results. Therefore, the use of non-preemptive scheduling is advocated.

In terms of timing analysis, the principal difference between preemptive and non-preemptive scheduling is the degree of blocking suffered. The blocking time caused by lower priority tasks for the non-preemptive model is expressed in equation (4.1). Equation (4.1) represents the maximum worst-case execution time for the set of tasks with lower priorities than task $i$. In comparison, blocking times with preemptive scheduling tend to be lower.

$$B_i = max_{k \in lp(i)}(C_k) \tag{4.1}$$

where $lp(i)$ is the set of lower priority tasks than task $i$

The overall equation for iteratively calculating task's worst-case response times with a non-preemptive scheduling model is represented in equation (4.2).

$$R_i^{n+1} = C_i + max_{k \in lp(i)}(C_k) + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{4.2}$$

with $R_i^0 = C_i + max_{k \in lp(i)}(C_k)$
which terminates when $R_i^{n+1} = R_i^n$, or $R_i^{n+1} > D_i$.
where $D_i$ is the deadline of task $i$

### 4.3.2 Optimality of the Deadline Monotonic Priority Ordering with Non-Preemptive Scheduling

Leung and Whitehead [61] shows that the DMPO approach is optimal for the preemptive computational model with the conditions that all tasks have zero offsets and deadlines less than or equal to their period. However, is DMPO optimal with the restricted computational model of non-preemptive scheduling?

**Theorem 1.** *DMPO is optimal for non-preemptive scheduling.*

**Proof**[1]

DMPO is optimal if for any task set, Q, that is schedulable by priority scheme, W, is also schedulable by DMPO. The proof of optimality of DMPO involves transforming the priorities of Q (as assigned by W) until the ordering is DMPO. With each step of the transformation schedulability is preserved.

Let $i$ and $j$ be two tasks with adjacent priorities in Q, such that under W: $P_i > P_j$ and $D_i > D_j$. $P_i$ is the priority of task $i$. Define scheme W' to be identical to W except that tasks $i$ and $j$ are swapped. Consider the conditions of schedulability of Q under W', with the reason given (in italics) why the condition is met:

1. All tasks with priorities greater than $P_i$ are unaffected by changes to lower priority tasks.

   *The non-preemptive scheduling model fulfills this constraint since the effect on schedulability is related to lower priority tasks.*

2. All tasks with priorities lower than $P_j$ are unaffected since they experience the same interference from tasks $i$ and $j$ irrespective of their priority order.

   *The set of tasks with priorities less than task $j$ remains unchanged and their impact remains the same, i.e. $B_i = max_{k \in lp(i)}(C_k)$. Therefore, this condition still holds.*

[1] This proof is adapted from a proof in Burns and Wellings [95] that dealt with the preemptive form of fixed priority scheduling.

3. Task $j$, which was schedulable under W, now has a higher priority, this means it suffers less interference and hence must be schedulable under W'.

   *This condition is true for the non-preemptive computational model because the higher priority tasks are not affected by the swapping of tasks' priorities. The blocking time for task $j$, $B_j$, could be greatly increased if the value of $C_i$ is large. However, the increase in the blocking time $B_j$ cannot be more than the decrease in the interference $I_j$ because under W task $i$ would have executed at least once.*

All that is left is the need to show task $i$, which has had its priority lowered, is still schedulable. Under W, it can be stated that $R_j \leq D_j$, $D_j < D_i$ and $D_i \leq T_i$. Therefore task $i$ interferes only once during the execution of $j$. Once the tasks have had their priorities switched, the new response time of task $i$ becomes equal to the old response time of task $j$. This is true because under both priority orderings, $C_j + C_i$ amount of computation time has been completed with the same level of interference from the higher priority tasks. Task $j$ is released only once during $[0, R_j]$, and hence interferes only once during the execution of task $i$ under W'. It follows that:

$$R'_i = R_j \leq D_j < D_i \tag{4.3}$$

Therefore it can be concluded that task $i$ is schedulable after the change of priorities. Priority scheme W' can now be transformed (to W") by choosing two more tasks that are in the wrong order according to DMPO and switching them. Each task switch preserves schedulability. Eventually, there are no more tasks to switch because the ordering is exactly that of DMPO and the task set is schedulable. Hence, DMPO is optimal for the general non-preemptive scheduling model. □

### 4.3.3 The Choice of Task Release Model

Section 4.3.1 states the general fixed priority scheduling model assumes sporadic tasks exist as well as periodic tasks. For similar reasons to the choice of non-preemptive control flow, the computational model that is primarily being considered is one featuring periodic tasks only. Sporadic tasks are excluded, not because

the timing analysis is hard but because the subsequent functional verification is made more difficult by the tasks' release time being unpredictable. However, most of the work to be presented is equally applicable to a computational model using sporadic tasks. Where it is not applicable an explicit statement is made.

### 4.3.4 Improved Blocking Model

A drawback of using a non-preemptive scheduling model, instead of a preemptive model, is that tasks tend to suffer greater blocking. In many systems, the blocking time can be prohibitive, especially when it is considered that the lower priority tasks are often the most computationally intensive. Therefore, an investigation is performed in this section with the aim of reducing any pessimism. The discussion contained within this section is purely related to the computational model that features just periodic tasks. Section 4.3.3 has already stated that the computational model could consist entirely of periodic tasks. If sporadic tasks are part of the task set, then the equations for blocking contained in equation (4.1) must be used.

It can be stated that for any tasks, which have:

1. a common critical instance (i.e. the tasks have the same offset) harmonic iteration rates, and

2. the worst-case response time of the lower priority task is less than or equal to the period of the higher priority task,

there is no blocking. Figure 4.1 is used to help illustrate that blocking doesn't occur in this case. Figure 4.1 presents the worst-case response time of two tasks $A$ and $B$, where task $A$ has an update rate of X and zero offset, and task $B$ has an update rate of 2X and zero offset. Figure 4.1 shows every second release of task A (i.e the task that can be blocked) is not blocked by task B. The reason is task B has already completed when task A is re-released. The shaded boxes represents the Worst-Case Response Time (WCRT) of a particular task release. Due to the fact many systems are evolved from systems previously using the

cyclic scheduler, there are likely to be a large number of tasks with harmonic iteration rates. The revised blocking model can be expressed by equation (4.4).

$$B_i = max_{\forall k}(C_k - \Delta) \tag{4.4}$$

where $\Delta$ is one clock cycle, and
the tasks in the set of tasks that can cause blocking satisfy
the following two conditions:

1. the task has a lower priority than task $i$, and
2. the task must either:
   have a non-identical release time to task $i$, or
   have a worst-case response time greater than the period of task $i$

A more formal representation of equation (4.4) is represented in equation (4.5). This representation is only included for completeness.

$$k \in lp(i), \forall n : \mathbb{N} | (T_k \neq nT_i \wedge T_i \neq nT_k) \vee O_k \neq O_i \vee R_k > T_i \tag{4.5}$$

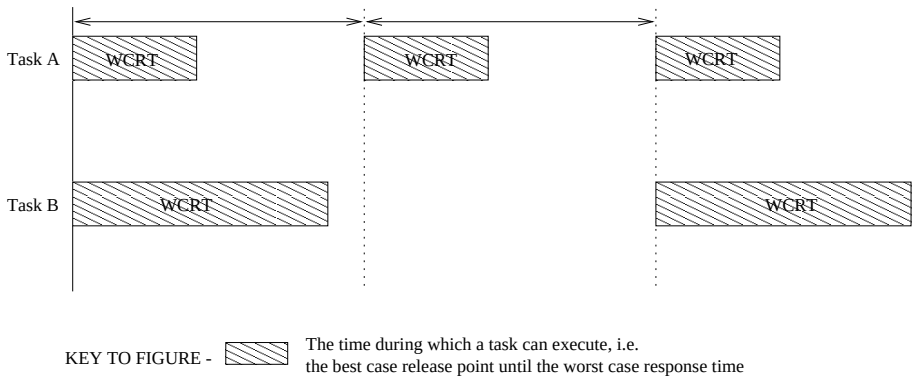


Figure 4.1: Illustration of Blocking Model Pessimism

To allow the improved blocking model to be proven correct, it is necessary to show it is a sufficient test.

**Theorem 2.** *A lower priority task, t, cannot block a task if the lower priority task is always released at the same time as the next instance of task t and it always completes before the next instance of task t.*

**Proof**

If a lower priority task (task $k$) is to cause blocking it must be executing when the task (task $i$) being analysed is released. In this case, blocking can only be avoided if the execution of the tasks $i$ and $k$ never overlaps. This can only be guaranteed if task $k$ is always released at the same time as task $i$. Therefore, the period of task $k$ must be an integer multiple of the period of task $i$, and task $k$ and $i$ must have the same period. This condition is also represented by equation (4.6).

$$(T_i = nT_k \vee mT_k = T_i) \wedge O_k = O_i, \text{ where } n, m \in \mathbb{N} \tag{4.6}$$

If task $k$ is to cause blocking, then it must be able to execute when task $i$ is released, therefore the condition in equation (4.7) must be true.

$$\text{i.e. } R_k > T_i \tag{4.7}$$

Therefore, for blocking not to occur the condition in equation (4.8) must hold.

$$\text{i.e. } R_k \leq T_i \tag{4.8}$$

For blocking to occur, either of the conditions in equation (4.6) or (4.8) must not arise.

□

Instead of the usual expression, $B_i = max_{\forall k}(C_k)$, equation (4.4) uses the term $B_i = max_{\forall k}(C_k - \Delta)$.

**Observation 1.** *The blocking time is one clock cycle less than the worst-case execution time of the task, k, causing the blocking.*

**Argument**

The reason the blocking time is one clock cycle less than the worst case execution time of task $k$ is illustrated in Figure 4.2. Figure 4.2 shows that both blocking or interference cannot occur after a task has started executing. Therefore, it can be stated that at least one clock cycle of task $k$ must already have occurred. This one clock cycle of task $k$ does not contribute to the blocking suffered by task $i$. That is task $k$ is always simultaneous released with task $i$, and task $k$

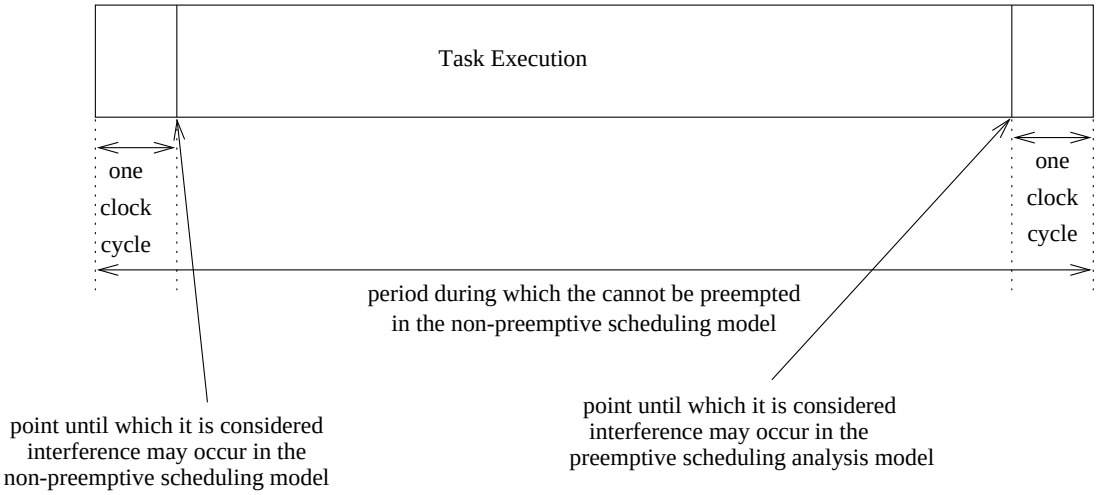


Figure 4.2: Task Executions to Illustrate the Edge Effect

always completes before task $i$ is re-released. Therefore, if the tasks execute for a shorter time, the blocking still does not occur. Hence, the blocking model in equation (4.4) is correct.

□

It should be noted that if the computational model does not feature offsets, then a simplified version of equation (4.5) given in equation (4.9) can be used.

$$k \in lp(i), \forall n \in \mathbb{N} | (T_k \neq nT_i \wedge T_i \neq nT_k) \vee R_k > T_i \tag{4.9}$$

Table 4.1 illustrates an example of how the improved blocking model increases the chance of schedulability. In Table 4.1, column $R_{old}$ is the results of the schedulability analysis with the old blocking model, whilst column $R_{new}$ is for the new blocking model. The table clearly shows the worst-case response time for Tasks $A$ and $B$ are reduced with the new approach.

| Id | T | C | D | P | $B_{old}$ | $R_{old}$ | $B_{new}$ | $R_{new}$ |
|---|---|---|---|---|---|---|---|---|
| A | 25 | 5 | 25 | 1 | 10 | 15 | 0 | 5 |
| B | 25 | 10 | 25 | 2 | 10 | 25 | 0 | 15 |
| C | 100 | 10 | 100 | 3 | 0 | 25 | 0 | 25 |

Table 4.1: Schedulability Results for the Improved Blocking Model

The columns In Table 4.1 represents:

$Id$ is the identifier of the task
$T$ is the period of the task
$C$ is the worst-case execution time of the task
$D$ is the deadline of the task
$P$ is the priority of the task
$B_{old}$ is the blocking time calculated using equation (4.1)
$B_{new}$ is the blocking time calculated using equation (4.4)
$R_{old}$ is the worst-case response time calculated using the value of $B_{old}$
$R_{new}$ is the worst-case response time calculated using the value of $B_{new}$

### 4.3.5 Improved Interference Model

If the schedulability analysis of section 3.1 is used, then another area where pessimism could be introduced is in the interference model. The purpose of this section is to investigate where the pessimism arises and how it may be reduced.

The interference term, given in equation (4.10), is the maximum amount of time higher priority tasks can execute before the task being analysed. The interference is calculated using equation (4.11). Therefore, the interference is the sum of the utilisations over the duration of interest of all the tasks with higher priority than task $i$. The utilisation is the product of the number of times the task can execute and its worst-case execution time. The number of times a higher priority task can execute is found by rounding up the result of the time during which interference may occur (i.e. the response time of the task being analysed) divided by the period of the higher priority task. Therefore, the interference can be calculated using equation (4.12).

$$I = \sum_{\forall j \in hp(i)} (\text{max. executions of task}_j) C_j \tag{4.10}$$

$$\text{max. executions of task}_j = \left\lceil \frac{\text{worst-case response time}_i}{\text{period of task}_j} \right\rceil = \left\lceil \frac{R_i}{T_j} \right\rceil \tag{4.11}$$

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{4.12}$$

where $hp(i)$ is the set of higher priority tasks than task $i$.

Equation (4.12) is clearly pessimistic, because the numerator of the interference term assumes a preemptive model. In a non-preemptive model, a higher priority task cannot commence execution if a lower priority task has begun executing. Instead, the higher priority task must wait for the lower priority task to complete. Therefore, the maximum number of executions of task $j$ can be improved as shown in equation (4.13). In this equation, the numerator is reduced by the worst-case execution time of task $i$, which is the task whose interference is being calculated. However, to prevent edge effects, where a task is released simultaneously as another is dispatched, then *one* clock cycle is added to prevent anomalies.

$$\text{max. executions of task}_j = \left\lceil \frac{R_i - C_i + \Delta}{T_j} \right\rceil \tag{4.13}$$

Figure 4.3 is used to help illustrate the impact of edge effects. In Figure 4.3 the boxes represent the worst-case response time of each task. Consider the time indicated by the vertical dotted line. Figure 4.3 shows a higher priority task (Task A) being released as a lower priority task (Task C) becomes the highest priority runnable task. Task C is the highest priority runnable task by virtue of the previous highest priority task (Task B) completing execution. Without the $\Delta$ term in equation (4.13), task C would be released. However in practice task A should be released. Therefore, the results would have been optimistic. With the $\Delta$ term, the correct worst-case execution sequence is maintained and the analysis is a sufficient form of timing analysis. The $\Delta$ term discussed here is the same as used in the improved blocking model.

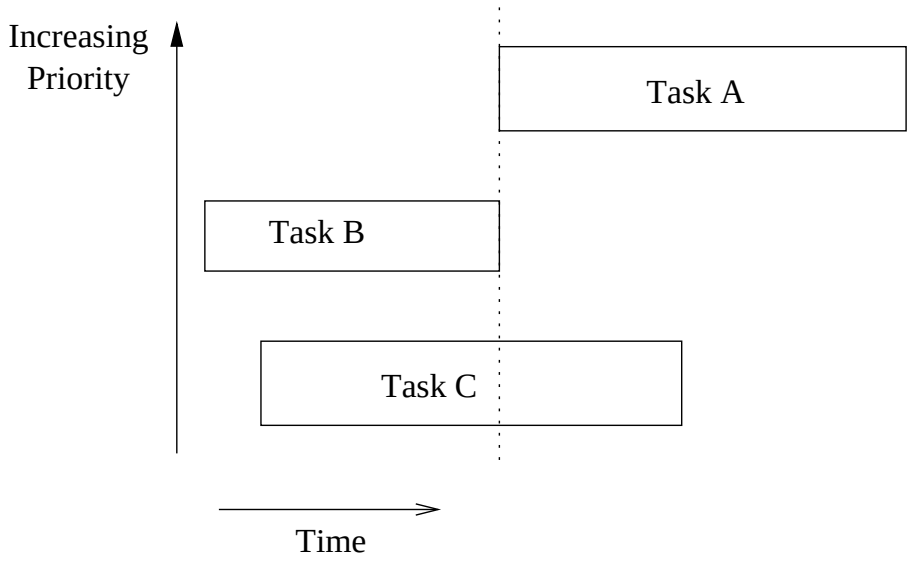


Figure 4.3: Task Executions to Illustrate the Edge Effect

Therefore, the revised interference equation is given in equation (4.14). The reduced value for the numerator can be significant if it means further instances of the higher priority tasks are not released.

$$I = \sum_{j \in hp(i)} \left\lceil \frac{R_i - C_i + \Delta}{T_j} \right\rceil C_j \tag{4.14}$$

### 4.3.6 Optimality of Deadline Monotonic Priority Ordering With The Restricted Computational Model

Section 4.3.2 shows that DMPO is optimal with the unrestricted computational model for non-preemptive scheduling. However, the computational model has been further restricted by only having periodic tasks. This has resulted in changes to the analysis. Therefore, the optimality of DMPO needs to be re-assessed under these conditions.

The change of computational model and associated analysis means the set of tasks that can cause blocking is now dependent on tasks' response times. Therefore, a change of priorities could cause more blocking. The following is an example that demonstrates DMPO is not optimal with this computational model. Consider the schedulability of the task set illustrated in Table 4.2 and 4.3, where the tasks in Table 4.2 are ordered using DMPO and the tasks in Table 4.3 have

| Id | T | D | C | B | P | R | Met? |
|---|---|---|---|---|---|---|---|
| A | 4 | 4 | 2 | 3 | 1 | 5 | N |
| B | 16 | 15 | 1 | 0 | 2 | 3 | Y |
| C | 16 | 16 | 4 | 0 | 3 | 7 | Y |

Table 4.2: Schedulability Results with DMPO

| Id | T | D | C | B | P | R | Met? |
|---|---|---|---|---|---|---|---|
| A | 4 | 4 | 2 | 2 | 1 | 4 | Y |
| B | 16 | 15 | 1 | 0 | 3 | 11 | Y |
| C | 16 | 16 | 4 | 0 | 2 | 6 | Y |

Table 4.3: Schedulability Results without DMPO

an alternative ordering. The columns have the same meaning as for Table 4.1, except for column *Met?* that represents whether the task is schedulable.

Table 4.2 shows the blocking caused by task C with DMPO leads to an unschedulable solution. However, Table 4.3 shows that if the priorities of tasks B and C are swapped, then the task set is schedulable. Hence, DMPO is sub-optimal for a purely periodic non-preemptive computational model. However, it is still optimal if the task set includes sporadic tasks.

It should be noted that in Table 4.3, the blocking time for task A is smaller, i.e. 2 rather than 3 units, than the value calculated with equation (4.4). The reason originates from performing exact analysis on the task set (refer to section 3.1.2 for details). The worst-case response of the tasks over the duration [0, Least Common Multiple of the Task Periods) for the priority orderings given in Tables 4.2 and 4.3 are illustrated in Figures 4.4 and 4.5 respectively.

In Figure 4.5, it is seen that the latest time task C can be dispatched is two clock cycles before task A is released for the second time. Therefore, instead of the blocking term being calculated based on $C_k - \Delta$, it is calculated using $C_k - 2\Delta$. This leads to a further improvement, given in equation (4.15). Equation (4.15) accounts for the reduced impact of task $k$ by nature of its worst case dispatch time.

$$B_i = max_{\forall k}(C_k - |R_k - T_i|_0) \tag{4.15}$$

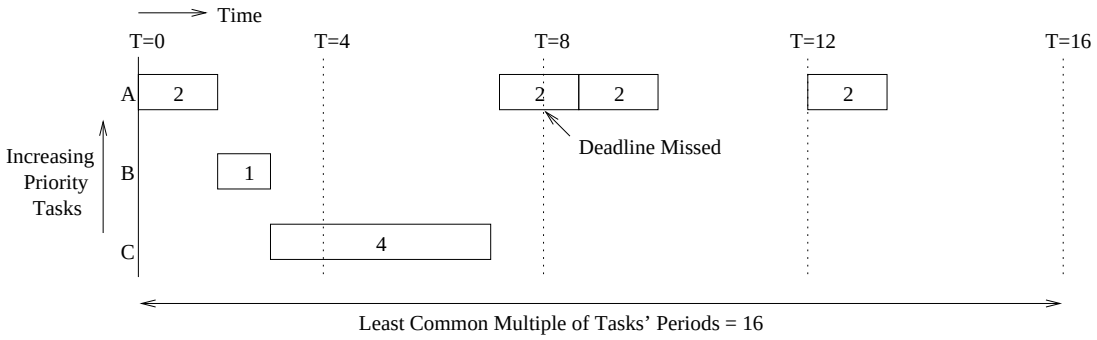


Figure 4.4: Worst-Case Task Execution for Deadline Monotonic Priority Ordering

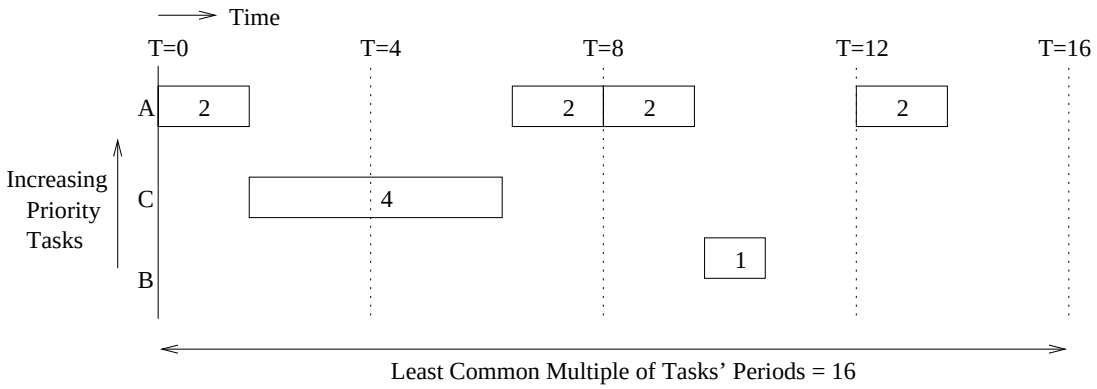


Figure 4.5: Worst-Case Task Execution for Another Priority Ordering

To support the four criteria for technology transfer, equations (4.4) and (4.15) should only be used when necessary (i.e. the task set's requirements are otherwise not met) because the original equation (4.1) is easier to understand.

## 4.4 Work That Remains for Fixed Priority Scheduling to Be a Complete Solution

Having considered the contents of chapters 2 and 3 as well as this chapter, it is not immediately apparent that the current theory can adequately support the system's timing requirements. This section raises two principal issues, which are the need to provide support for fixed priority scheduling in the infrastructure and the need to assign attributes to tasks.

### 4.4.1 Infrastructure Support for Fixed Priority Scheduling

An issue to be considered is how a fixed priority scheduler interacts with the infrastructure of the system. The main parts of the infrastructure of interest are the scheduling mechanism and the timing overrun detection mechanism. Chapter 5 is to investigate how a fixed priority scheduler may interact with an existing infrastructure left from a cyclic scheduling application. Part of this work is to derive effective timing analysis. The solution derived must consider the four criteria for successful technology transfer, particularly the reuse criterion.

### 4.4.2 The Need for Task Attribute Assignment

Most of the current scheduling work assumes a task has a well specified period and deadline that enables the system's timing requirements to be met. However in practice, one of the key roles of the system designer is to actually calculate what the deadlines need to be, in order to meet the system's timing requirements. In particular, cyclic scheduler makes no use of deadlines whereas fixed priority scheduling relies on the use of deadlines for the purposes of analysis and synthesis. Therefore task deadlines need to be defined.

Many academic texts assume that if the task's attributes are not known then simply setting the task's deadline to equal its period is sufficient. However, this does not allow for the needs of transactions and tasks with jitter requirements. Priority inheritance could be proposed as the solution to needing transactions. Chapter 3 explains how priority inheritance is not an ideal technique for solving the problem. Therefore, the challenge is to assign the task's attributes so the timing requirements in section 2.1 can be met.

Chapter 6 investigates how the system's timing requirements can be met based on the infrastructure proposed in Chapter 5 and using the deadline monotonic priority ordering. A perceived disadvantage of fixed priority scheduling is a lack of determinism. For certification purposes, it is useful for the developer to be able to easily justify the requirements are met. Therefore, any approach derived should support this need.

## 4.5 Summary

This chapter has achieved four aims. These are:

1. how a fixed priority scheduler can solve the problems of the cyclic scheduler,
2. how the problems of a fixed priority scheduler can be mitigated against,
3. how an effective move to cyclic scheduling can be achieved, and
4. the work that is necessary to support the domain and application.

The chapter has also provided improvements to the "standard" schedulability analysis, taken from section 3.1, for a restricted computational model. The restricted computational model is all tasks are periodic and executed non-preemptively. The technique also compares favourably with the criteria for successful technology transfer.

1. *Certification* - The approach provides analysis that guarantees the system's timing behaviour. Therefore, the results of the analysis can be used as part of the certification evidence of the system.

2. *Reuse* - The greatest advantage of using the proposed non-preemptive approach is the high-level of reuse that is afforded. Any changes to the software can be limited to the scheduler module alone, i.e. there is no need to change any of the application software.

3. *Understanding* - The satisfaction of the criterion is demonstrated by the fact the approach has been technology transferred to industry, where it has been successfully applied [96]. Further evidence is presented in the case study in Chapter 7.

4. *Sufficiency* - The approach facilitates better support for a more flexible computational model than the cyclic scheduler. Therefore, the sufficiency criterion is satisfied.

The chapter has identified two areas of work to support the uniprocessor solution for a fixed priority scheduler, which are:

1. how to modify the infrastructure left from a cyclic scheduler based system for use with a fixed priority scheduler including the appropriate timing analysis (dealt with in Chapter 5), and

2. how to derive task attributes so that the timing requirements are met (dealt with in Chapter 6).

# Chapter 5

# Infrastructure Choice and Associated Timing Analysis

This chapter presents an investigation of how the existing infrastructure, left over from the cyclic scheduler, may be utilised after a transition to fixed priority scheduling, and how the resultant overheads are analysed. The starting position for considering the transition is the model for the cyclic scheduler discussed in section 2.5. When deciding what changes are to be made to the infrastructure, the four criteria defined in section 1.3 (certification, sufficiency, understanding, and reuse) for successful technology transfer must be considered.

The aim of the work is to cause the minimum possible change to the way the system operates, whilst still harnessing the advantages of fixed priority scheduling. The chapter is split into two parts, which are: how the task release mechanism should be implemented (in section 5.1), and how timing overruns should be detected (in section 5.2). The two parts basically define how the scheduler is to work. Part of the work that is key to the first two parts is analysing the overheads of the scheduler. The analysis of a scheduler's overheads is an area of work that has received very little attention. The majority of academic work assumes an ideal mechanism for handling tasks, where the highest priority task is always executed, overheads are zero and faults do not occur.

## 5.1 Implementing and Analysing the Task Release Mechanism

The purpose of the task release mechanism is to release tasks in an effective manner that allows the timing requirements to be met. This section investigates how best to release tasks. There are three criteria considered; the need to reuse as much of the system from the cyclic scheduler (and its applications) as possible, the need to keep kernel overheads low, and the need for a responsive system. The latter two criteria are derived from the sufficiency criterion of section 1.3.

### 5.1.1 Tick Driven Task Release

Section 2.5 states that in the cyclic scheduler model there is a regular clock tick that triggers a sequence of tasks corresponding to a particular minor cycle. The fixed priority scheduler could be implemented using the available clock tick resulting in virtually no change in the system operation. Each time a clock tick occurred, a decision would be taken of whether tasks should be released. Between ticks, runnable tasks would be dispatched in priority order. In a non-preemptive scheduler, when the clock tick occurs the task executing could be interrupted to allow the run queue to be updated. In this case, when the updating of the run queue is finished, the interrupted task is resumed.

Section 3.1.9 of the literature survey explains how the clock tick may be accounted for in the analysis. Two approaches are presented, one based on a single task and the other based on multiple tasks. There is a trade-off between simplicity and pessimism of the single and multiple task approaches. The multiple task approach offers greater accuracy with respect to response times. However, the multiple task approach requires more information to be obtained. It should be noted that the key points made in this chapter are independent of the way the overheads due to the clock tick are modelled.

The problem with the tick driven approach is that tasks not released at a harmonic rate of the clock tick suffer from release jitter. These tasks can have a release jitter as great as the period of the clock tick, since the task could be due for release an instant after the current clock tick. Audsley [54] states that for a

periodic task the release jitter is given by equation (5.1). Equations (3.5) and (3.6) in section 3.1 shows how release jitter is accounted for in the schedulability analysis.

$$J_i = T_{clk} - gcd(T_{clk}, T_i) \tag{5.1}$$

where $gcd$ is the greatest common divisor, and
$T_{clk}$ is the clock tick rate

If the release jitter becomes too great, then the task iteration rates or the clock tick rate may have to be altered so that task deadlines can be met. However, a significant problem with cyclic scheduling caused by the inflexibility of the computational model is having to change task rates to suit the infrastructure. The option of altering task iteration rates so that the release jitter is reduced effectively constrains the task's iteration rates to harmonics of the clock tick rate. If this strategy is adopted, then one of the benefits of fixed priority scheduling is effectively lost.

The alternative is changing the clock tick period. To eliminate the effects of release jitter would need the term $gcd(T_{clk}, T_i)$ in equation (5.1) to be equal to $T_{clk}$ for all tasks in the task set. Therefore, the condition in equation (5.2) would have to be satisfied.

$$\forall i \in \text{Tasks} : gcd(T_i, T_{clk}) = T_{clk} \tag{5.2}$$

The problem with altering the clock tick period is the period becomes dependent on tasks' periods. Therefore, the clock tick period would be prone to change and could be quite small. The consequence of $T_{clk}$ being small is that the overheads in the system would be significantly increased. Equation (5.3) gives the overheads caused by the clock tick mechanism. The equation is based on equation (3.11). It represents the worst-case execution time of the function performing the tick driven release mechanism, multiplied by the number of times it is performed over the duration of interest, i.e. the least common multiple of the tasks' periods. The value of $T_{clk}$ should be chosen so that the minimum value for the overhead is obtained.

$$U_{overhead} = \frac{LCM(T_j)}{T_{clk}}(MC_{first} + (\#j - M)C_{sub}) \tag{5.3}$$

where $U_{overhead}$ is the utilisation due to the overhead of the clock tick,
$C_{first}$ is the cost of releasing the first task,
$C_{sub}$ is the cost of releasing the subsequent tasks,
$j$ is a task in the set of tasks to be executed,
$T_{clk}$ is the clock tick rate, which is constrained by the condition in equation (5.2), i.e. all tasks in the task set have a period that is an integer multiple of the clock tick rate
$\#j$ is the number of tasks in the task set, and
$LCM(T_j)$ is the least common multiple of the periods of the tasks in the task set

**Observation 2.** *The best value of the clock tick period is found by minimising the utilisation of the clock overheads.*

**Argument**

Solving equation (5.3) provides the value of $T_{clk}$ such that the utilisation of the clock overheads is minimised over the time the tasks take to repeat, i.e. the least common multiple of the clock periods. Reducing the utilisation of the overheads to a minimum is considered the best solution, but not optimal. The reason it is not optimal is a particular phasing of tasks could mean minimal overheads are needed during a particular time frame rather than in general.

□

### 5.1.2 Time Driven Release

An alternative approach to releasing tasks is the time driven approach. The time driven approach releases tasks dependent on a real-time clock rather than a regular clock tick. When a task finishes executing, the clock is read to allow the run queue to be updated with tasks that should be released. Only then is the highest priority task dispatched.

For further discussion of the time driven release mechanism and its analysis, refer to section 3.1.9 of the literature survey. The following section compares the tick driven and time driven approaches.

### 5.1.3 Comparison of the Tick Driven and Time Driven Task Release Mechanisms

The task release mechanism chosen can impact three principal areas; the hardware architecture, kernel overheads and responsiveness:

1. *Hardware Architecture*

   The infrastructure of the system is affected by the scheduling policy. The tick driven scheduler requires a hardware clock tick, whereas the time driven scheduler requires a real-time clock. The advantage of the tick driven approach is that a system previously based on a cyclic scheduler already has hardware to generate a clock tick that can be reused. It should be noted that an assumption is made that the clock tick rates available are completely flexible. In practice it may not be, leading to design changes to reduce release jitter. Therefore, the reuse advantage is reduced.

2. *Kernel Overheads*

   Even though kernel overheads are normally relatively small in comparison to the overall resource available, the overheads can still cause scheduling problems. It is all too easy for the kernel to be implemented in an inefficient manner taking excessive resources, or to take valuable resources at critical times, or for the worst-case (theoretical) analysis to be much greater than the actual worst case. In general, the task release mechanism that makes the most attempts to update the run queue has the largest kernel overheads. The reason is that an overhead is incurred independent of whether a task is released. This assumes both mechanisms have a similar overhead searching for the highest priority runnable task and subsequently releasing the task that is found. The assumption is quite realistic.

3. *Responsiveness*

   The responsiveness (i.e. the ability to meet deadlines) is related to the release jitter and the blocking time. The time driven approach is considerably

more responsive, since tasks are released as soon as the currently executing task is complete. The execution time of this task is already allowed for in the analysis through either the blocking term or the interference term. Therefore, no additional release jitter is introduced.

There are good and bad points associated with both methods discussed for releasing tasks. Rather than try to decide which technique is best, it is easier to say that neither approach is ideal in all circumstances and attempt to derive a better solution.

### 5.1.4 A Hybrid Approach to Task Release

An alternative task release mechanism proposed is a hybrid of the tick driven and time driven scheduling approaches. The hybrid approach releases the majority of tasks based on a clock tick, with a few carefully selected tasks released in a time driven manner. The tasks released by the time driven approach are those whose rate is not a harmonic of the clock tick rate.

The benefit of the hybrid approach is that it allows a compromise between kernel overheads and task responsiveness with the minimum change to the task release mechanism from the cyclic scheduler. The kernel overheads are reduced since between clock ticks only those tasks requiring a quick response need to be checked for release. Therefore each time a task finishes executing, the kernel overheads each time a task finishes executing is reduced when compared with the time driven approach. Responsiveness is improved because tasks not released at a harmonic of the clock tick period do not suffer any unnecessary release jitter.

The actual clock tick period could be chosen to minimise the amount of overheads. The clock overheads are represented in equation (5.4). Equation (5.4) is derived from equations (5.3) and (3.13) It should be remembered that in some implementations the clock tick mechanism may not be re-programmable, or there may be a limited selection of iteration rates available for the clock tick. The value of $T_{clk}$ that minimises the overheads is considered the best. However, it cannot be claimed as optimal because a particular phasing of tasks could exist so that

| Task | T | C | D |
|---|---|---|---|
| A | 6250 | 250 | 6250 |
| J | 11000 | 1000 | 11000 |
| B | 25000 | 4000 | 25000 |
| C | 50000 | 2000 | 50000 |
| D | 100000 | 1000 | 100000 |
| E | 200000 | 1000 | 200000 |
| F | 1000000 | 3000 | 1000000 |

Table 5.1: Basic Task Set

it would be better to have the overheads occur at specific times rather than minimise their magnitude.

$$\begin{aligned} U_{overhead} &= \text{Overheads due to the tick driven release mechanism} + \\ &\quad \text{Overheads due to the time driven release mechanism} \end{aligned}$$

$$\begin{aligned} U_{overhead} &= \left( \frac{LCM(T_l)}{T_{clk}} \left( MC_{first} + (\#j - M)C_{sub} \right) \right) + \\ &\quad \frac{LCM(T_l)}{T_k} C_{COOP} \end{aligned} \tag{5.4}$$

where $C_{COOP}$ is the cost of releasing a task in a time driven fashion,
$j$ is the set of tasks released in a tick driven manner,
$k$ is the set of tasks released in a time driven manner,
$l$ is the set of tasks, and
$LCM(T_l)$ is the least common multiple of the tasks' periods

### 5.1.5 An Example To Illustrate The Different Forms of Task Release Analysis

This section presents an example to demonstrate how the hybrid mechanism is applied and its associated analysis. Table 5.1 presents the task set that is to be scheduled and analysed.

Initially, the system is analysed assuming a tick driven approach using a single task to model clock overheads. The task set of Table 5.1 is extended to include a

| ID | P | T | C | D | B | J | R | Met? |
|---|---|---|---|---|---|---|---|---|
| $clk$ | 1 | 6250 | 2000 | 6250 | 4000 | 0 | 6000 | Yes |
| A | 2 | 6250 | 250 | 6250 | 4000 | 0 | 6250 | Yes |
| J | 3 | 11000 | 1000 | 11000 | 4000 | 6000 | 15500 | No |
| B | 4 | 25000 | 4000 | 25000 | 0 | 0 | 12750 | Yes |
| C | 5 | 50000 | 2000 | 50000 | 0 | 0 | 15750 | Yes |
| D | 6 | 100000 | 1000 | 100000 | 0 | 0 | 19000 | Yes |
| E | 7 | 200000 | 1000 | 200000 | 0 | 0 | 20000 | Yes |
| F | 8 | 1000000 | 3000 | 1000000 | 0 | 0 | 23000 | Yes |

Table 5.2: Analysis for a Tick Driven Scheduler, where $T_{clk} = 6250$, with a Single Task to Model Overheads

high priority task, task $clk$, that represents all the overheads due to the periodic clock. According to the single task model, $clk$ has the following characteristics (based on the parameters $C_{first} = 500$ and $C_{sub} = 250$):

$$\begin{aligned} T_{clk} &= D_{clk} = 6250 \\ \therefore C_{clk} &= C_{first} + (N - \left\lceil \frac{T_{min}}{T_{clk}} \right\rceil) C_{sub} = 500 + (7 - \left\lceil \frac{6250}{6250} \right\rceil) 250 = 2000 \end{aligned} \quad (5.5)$$

The results of the schedulability analysis are given in Table 5.2, which shows task $J$ misses its deadline. Another choice for the period of task $clk$ could be 25000. However, if this value is chosen then the release jitter for task $J$ would be 6250. Since the jitter would be equal to the deadline, then the task set is clearly unschedulable. To eliminate release jitter, based on equation (5.2) the value of $T_{clk}$ would have to be 250. The fact that $T_{clk}$ would be less than $C_{first}$ means the overheads use all the processors resources resulting in an unschedulable task set.

The system is now analysed assuming tick driven scheduling using multiple tasks to model clock overheads, in order to investigate whether better results are obtained. The task set of Table 5.1 is analysed with each task having its own task to model clock overheads (e.g. task A has an associated task $clkA$).

The worst-case computation time of $clkA$ is given by $C_{first} = 500$. All other extra tasks ($clkB \cdot \cdot clkF$) can only be released at the same time as task $clkA$ since their periods are multiples of $T_{clkA}$. Hence, tasks $clkB \cdot \cdot clkF$ all have

| ID | P | T | C | D | B | J | R | Met? |
|---|---|---|---|---|---|---|---|---|
| $clkA$ | 1 | 6250 | 500 | 6250 | 0 | 0 | 500 | Yes |
| $clkJ$ | 2 | 11000 | 250 | 11000 | 0 | 6000 | 6750 | Yes |
| $clkB$ | 3 | 25000 | 250 | 25000 | 0 | 0 | 1000 | Yes |
| $clkC$ | 4 | 50000 | 250 | 50000 | 0 | 0 | 1250 | Yes |
| $clkD$ | 5 | 100000 | 250 | 100000 | 0 | 0 | 1500 | Yes |
| $clkE$ | 6 | 200000 | 250 | 200000 | 0 | 0 | 1750 | Yes |
| $clkF$ | 7 | 1000000 | 250 | 1000000 | 0 | 0 | 2000 | Yes |
| A | 8 | 6250 | 250 | 6250 | 4000 | 0 | 7000 | No |
| J | 9 | 11000 | 1000 | 11000 | 4000 | 6000 | 14250 | No |
| B | 10 | 25000 | 4000 | 25000 | 0 | 0 | 9250 | Yes |
| C | 11 | 50000 | 2000 | 50000 | 0 | 0 | 12000 | Yes |
| D | 12 | 100000 | 1000 | 100000 | 0 | 0 | 13000 | Yes |
| E | 13 | 200000 | 1000 | 100000 | 0 | 0 | 15250 | Yes |
| F | 14 | 1000000 | 3000 | 1000000 | 0 | 0 | 18250 | Yes |

Table 5.3: Analysis for a Tick Driven Scheduler, where $T_{clk} = 6250$, with Multiple Tasks to Model Overheads

computation times equal to $C_{sub}$. The results of the analysis are given in Table 5.3. In this case, tasks $A$ and $J$ miss their deadlines. Therefore, the results appear to be worse than with the single task model.

The analysis is repeated for the task set being scheduled using the time driven approach. Initially, the computation times of the tasks (as given in Table 5.1) are increased to account for the cost of updating the run queue using equation (3.13). It is assumed that $C_{coop} = C_{first} = 500$. It should be noted there is no release jitter with the time driven approach. Table 5.4 gives the results of the analysis, which shows all the tasks meet their deadlines. The reason the time driven approach produces better results is that the kernel overheads coincide with when tasks are released, rather than all the overheads occurring at the critical instant.

The hybrid approach is now considered. The clock rate is assumed to be 25000 as left from the cyclic scheduler. To eliminate jitter, tasks $A$ and $J$ are scheduled in a time driven fashion and the remainder of the tasks are tick driven. A single task, $clk$, with the highest priority is used to account for all tick driven overheads.

| ID | P | T | C | D | B | R | Met? |
|---|---|---|---|---|---|---|---|
| A | 1 | 25000 | 750 | 25000 | 4500 | 5250 | Yes |
| J | 2 | 11000 | 1500 | 11000 | 4500 | 7500 | Yes |
| B | 3 | 25000 | 4500 | 25000 | 0 | 7500 | Yes |
| C | 4 | 50000 | 2500 | 50000 | 0 | 12250 | Yes |
| D | 5 | 100000 | 1500 | 100000 | 0 | 13750 | Yes |
| E | 6 | 200000 | 1500 | 200000 | 0 | 15250 | Yes |
| F | 7 | 1000000 | 3500 | 1000000 | 0 | 18750 | Yes |

Table 5.4: Analysis for a Time Driven Scheduler

| ID | P | T | C | D | B | R | Met? |
|---|---|---|---|---|---|---|---|
| clk | 1 | 25000 | 1500 | 25000 | 0 | 1500 | Yes |
| A | 2 | 6250 | 750 | 6250 | 4000 | 6250 | Yes |
| J | 3 | 11000 | 1500 | 11000 | 4000 | 8500 | Yes |
| B | 4 | 25000 | 4000 | 25000 | 0 | 10750 | Yes |
| C | 5 | 50000 | 2000 | 50000 | 0 | 12750 | Yes |
| D | 6 | 100000 | 1000 | 100000 | 0 | 13750 | Yes |
| E | 7 | 200000 | 1000 | 200000 | 0 | 14750 | Yes |
| F | 8 | 1000000 | 3000 | 1000000 | 0 | 17750 | Yes |

Table 5.5: Analysis for a Hybrid Scheduler, where $T_{clk} = 25000$ with a Single Task to Model Overheads

It has parameters:

$$\begin{aligned} T_{clk} &= D_{clk} = 25000 \\ \therefore C_{clk} &= C_{first} + (N - \left\lceil \frac{T_{min}}{T_{clk}} \right\rceil) C_{sub} = 500 + (5 - \left\lceil \frac{25000}{25000} \right\rceil) 250 = 1500 \end{aligned} \tag{5.6}$$

The computation times of tasks $A$ and $J$ are increased (over their values in Table 5.1) to include the overheads for time driven scheduling, i.e. $C_{coop}$. The results of the analysis are given in Table 5.5, which shows all the tasks are schedulable.

From Tables 5.4 and 5.5, it can be seen the response times of some tasks are better with the hybrid approach than the time driven approach. The hybrid

release mechanism provides better worst-case response times than the time driven approach when the period of the task in question is greater than the period of the clock tick. Otherwise, the time driven approach provides the better response times. The reason for this is, in general the hybrid scheduling approach causes less overheads than the time driven approach. However, the hybrid approach phases part of its overheads (that associated with the task $clk$) at the critical instant, this causes more impact on the tasks with a shorter period than the clock tick. Since the majority of tasks should tend to have a period greater than the clock tick period, then this should not cause too great a problem. Therefore, hybrid scheduling provides a useful alternative to time driven scheduling. This is particularly the case where reuse from systems that previously used a cyclic scheduler is important.

### 5.1.6 Summary

This section has clearly shown how the hybrid approach may be used to release tasks effectively. The advantage of the hybrid approach over the tick driven approach is the flexibility to reduce the effects of jitter and minimise overheads, whilst largely maintaining the existing clock tick architecture. Whilst it is recognised that one example does not prove the strategy to be better, the overall philosophy of reducing jitter and overheads can only enhance the schedulability of any task set. The advantage of the hybrid approach over the time driven approach is that no change occurs in the way the majority of tasks are handled by the infrastructure following the transition from cyclic to fixed priority scheduling.

Based on the four criteria for acceptability of change, the hybrid approach is deemed successful.

1. *Reuse* - The existing tick driven architecture is reused except where the schedulability of the system dictates a change to a time driven approach is necessary. Therefore, only the minimum amount of change is necessary to the way in which tasks interact with the infrastructure. The key problem with the hybrid scheduling approach would arise if the system has no real-time clock. This would mean some of the tasks would have to be released have a period that is not a multiple of the clock tick rate. In this circumstance, the options to eliminate release jitter include:

(a) alter the task's period (which leads to an inflexible computational model),

(b) alter the clock tick rate (which may not be possible or may affect other tasks), or

(c) provide a real-time clock (which means the infrastructure is no longer completely reused).

Clearly, none of these options are ideal, and the developer has to make the decision based on the particular system.

2. *Sufficiency* - The amount of overheads is minimised and where necessary release jitter is eliminated, which increases the chance of schedulability. No artificial restrictions on iteration rates are enforced.

3. *Certification* - Verification is produced that gives a definitive statement of whether the system is schedulable, which is important evidence for the certification case.

4. *Understanding* - Both the tick driven and time driven approaches are considered simple to understand. With the hybrid approach, the decision of which tasks are released by each technique is straightforward. Only the tasks with an iteration rate that is not a multiple of the clock tick rate are released by the time driven approach.

## 5.2 Handling Timing Overruns

Section 2.2.1 justifies how a successful certification argument needs to demonstrate that timing overruns are detected within bounded time. Timing overruns may be caused by faults within the system, or by the incorrect application of analysis. The integrity of the timing overrun detection mechanism is particularly important for safety critical systems. The mechanism has to deal with both random and systematic failures.

With the cyclic scheduler, failures are detected by testing whether a task belonging to a minor cycle is executing when the clock tick arrives, which is an indication that an error has occurred. The timing watchdog is also used to

detect systematic failures caused by the difficulty in synthesising the schedule and the inexact methods used to estimate worst-case execution times. The importance of detecting timing overruns in the cyclic scheduling model has lead to high-integrity trusted watchdogs being developed. Section 2.4 provides an overview of how timing watchdogs may work in the cyclic scheduling model.

One of the key advantages of fixed priority scheduling is viewed as the graceful degradation of the system in the event of timing overruns [6]. The effect of a timing overrun is that some tasks may miss their deadline. Since higher priority tasks always have the first chance to execute, it is likely that the lower priority tasks are the ones that miss their deadline. Missed deadlines could be tolerated in cases where only the less critical tasks are affected. In these cases, the system is considered to degrade gracefully.

In safety critical systems rather than go to the effort of trying to establish what type of faults can be allowed, a frequently taken strategy is to assume that any fault has to be recoverable. A technique often adopted is to reset the particular lane where the fault occurred and change the lane controlling the system. A lane is considered to be the processor executing the software and the associated peripheral hardware, including memory and timing watchdog. The strategy assumes safety critical systems have active replication to support fault tolerance. However, some benefit could be obtained by allowing a lane to attempt fault recovery without reseting. There are a number of recognised techniques of fault recovery, including recovery blocks [97] and check-pointing [98].

Independent of the fault tolerance strategy used, there is still a need to recognise failures using a timing watchdog. The main requirement for the timing watchdog is to identify timing overruns, enabling the functionality for fault tolerance to assess what action is needed. There are two principal approaches for providing protection against timing overruns, which are the tick driven approach and the countdown timer approach.

### 5.2.1 Tick Driven Watchdog

The tick driven watchdog approach is where at a regular rate (normally a multiple of the clock tick rate) a check is performed to ensure that the software is not exceeding its time bounds. With a fixed priority scheduling approach, the check

could be based on ensuring that a sufficient number of tasks have been executed between checks. The check would assess whether the sum of the worst-case execution times of the tasks that have been executed between clock ticks is larger than the period of the timing watchdog. An alternative method is to check that all tasks that have to execute between clock ticks have done so . The tasks that have to execute are those with a period and deadline, less than, or equal to the clock tick rate. The response time of the tick driven watchdog to detectable faults is twice the period of the timing watchdog. The reason is the failure may not be detected the first time the timing watchdog executes but it will the second time.

The overheads may be accounted for by creating a new task with period equal to the clock tick and an appropriate worst-case execution time derived through analysis. The clock tick has to interrupt a task's execution in a preemptive fashion. Otherwise, a failed task could continue to execute forever. If the timing watchdog check has completed successfully, then the interrupted task resumes immediately. The preemptive nature of the timing watchdog means it should not be modelled as a conventional task according to a non-preemptive model. Therefore, the schedulability analysis should allow for the interference of the timing watchdog being preemptive as shown in equations (5.7) and (5.8).

$$R_i = C_i + B_i + I_i + I_{TW} \tag{5.7}$$

$$I_{TW} = \left\lceil \frac{R_i}{T_{TW}} \right\rceil C_{TW} \tag{5.8}$$

where $I_{TW}$ is the interference due to the timing watchdog software, and

$C_{TW}$ is the worst-case execution time of the timing watchdog software.

### 5.2.2 Countdown Timer Watchdog

The countdown timer watchdog approach is where each time a task commences execution, a countdown timer is started. The duration of the countdown timer is greater than the worst-case execution time of any task in the task set. When the task execution is complete, the countdown timer is restarted. If the countdown timer reaches zero, then fault recovery is performed. It is assumed that if a

task executes for longer than its worst case execution time, then a failure has occurred. Therefore, the response time to a fault is equal to the duration of the countdown timer.

The overheads of the countdown timer approach may be accounted for by increasing the worst-case execution time of all tasks as shown in equation (5.9).

$$C_i := C_i + C_{TW} \tag{5.9}$$

### 5.2.3 Comparison of the Timing Watchdog Approaches

The timing watchdog approaches can be assessed with respect to the same three parameters as the task release mechanism, which are; hardware architecture, kernel overheads, and responsiveness.

1. *Hardware Architecture* - There is no clear answer to which approach is better from a hardware architecture perspective. On the one hand, the tick driven approach has the obvious advantage that the existing architecture can be reused, which reduces the amount of rework. On the other hand, the countdown timer approach has the advantage that it does not use the existing architecture - this seems to be a contradiction. However, the countdown timer approach only uses an interrupt when a fault has occurred. If the countdown timer watchdog is combined with the time driven task release mechanism, then it should be possible to remove all interrupts from the system - at least in fault-free conditions. This can be a significant advantages for certification because engineers often struggle to justify the need for, and the safety of, interrupts in the system.

2. *Kernel Overheads* - Clearly, the countdown timer approach should cause greater interference due to the fact the watchdog software is executed more times. This assumes the two checks have a similar overhead each time they are performed. However, the overhead of both approaches is likely to be relatively small in relation to the system's processing resource.

3. *Responsiveness* - The countdown timer approach is considered to be more responsive for two reasons. Firstly, the countdown timer approach detects an overrun within the bounded amount of time set by the duration of the

timer. Whereas the tick driven approach could take as long as two clock periods to detect an overrun. Secondly, the countdown timer approach has the obvious advantage that the specific task causing the overrun can be identified, which cannot be guaranteed with the tick driven approach. Therefore, intelligent fault tolerance is only possible with the countdown timer approach.

The choice of watchdog mechanism can also be considered with respect to the four criteria for a successful transfer of technology.

1. *Reuse* - Obviously making use of the existing tick driven watchdog mechanism is better from the reuse perspective than having to develop and certify a new timing watchdog mechanism.

2. *Certification* - Both mechanisms satisfy the safety goal of detecting timing overruns. The tick driven watchdog has the advantage that it is already trusted, whereas the countdown timer watchdog has the advantage of better fault identification facilitating accurate diagnosis and correction. Also, the removal of interrupts from the system makes certification more straightforward.

3. *Sufficiency* - For systems where quicker or more accurate response to faults is necessary to provide intelligent fault tolerance or maintenance, the countdown timer driven approach is a clear winner.

4. *Understanding* - Both mechanisms can be considered simple to understand, implement and change. The satisfaction of the criterion is demonstrated by the fact the approach has been transferred to industry, where it has been successfully applied [96]. Further evidence is presented in the case study in Chapter 7.

## 5.3 Summary

This chapter has addressed two issues, how an infrastructure left over from a cyclic scheduler can support fixed priority scheduling and how the effects of the infrastructure can be analysed effectively. There are two primary parts to the work, which are the consideration of the task release mechanism, and the detection of timing overruns.

To improve the ability to meet the timing requirements of the system, i.e. the sufficiency criterion, a hybrid scheduler is derived for task release. The hybrid scheduler combines the best features of a tick driven and time driven task release. The hybrid scheduler allows the maximum reuse to be obtained, whilst not restricting the system's computational model.

The good and bad points of the timing watchdog mechanism are discussed leading to a conclusion that the best technique depends on the system's requirements. A tick driven approach is better where reuse is paramount. However a countdown timer approach may ease the certification effort and also supports intelligent fault management.

# Chapter 6

# Task Attribute Assignment

The purpose of this chapter is to investigate the issues related to task attribute assignment on an individual processor. The majority of work on fixed priority scheduling makes the assumption that tasks have their attributes pre-assigned, making priority assignment[1] trivial. In academic papers, the task attributes are normally considered to be period and priority. However, in practice deadlines and offsets must also be considered.

A significant challenge is to derive task attributes that meet the system's timing requirements in a way that can be understood by a non-specialist. An approach is proposed for task attribute assignment that caters for all the likely timing requirements of complex control systems imposed on the scheduler. In section 2.1.4, the timing requirements were summarised as:

1. *Task* - period, deadline, jitter, separation
2. *Transaction* - precedence of tasks, period, deadline, jitter

A secondary aim is to try to ensure that only one set of analysis (i.e. task schedulability analysis) verifies all the timing characteristics of the system with the benefit of the that a separate verification tool is not needed for transaction timing requirements.

[1]Priority assignment is the process of assigning priorities so that the timing requirements are met. Whereas, deadline assignment is the process of assigning priorities so that the timing requirements are met. However, for this work it is assumed tasks' priorities are derived from their deadlines using the DMPO - the shorter the deadline the higher the priority.

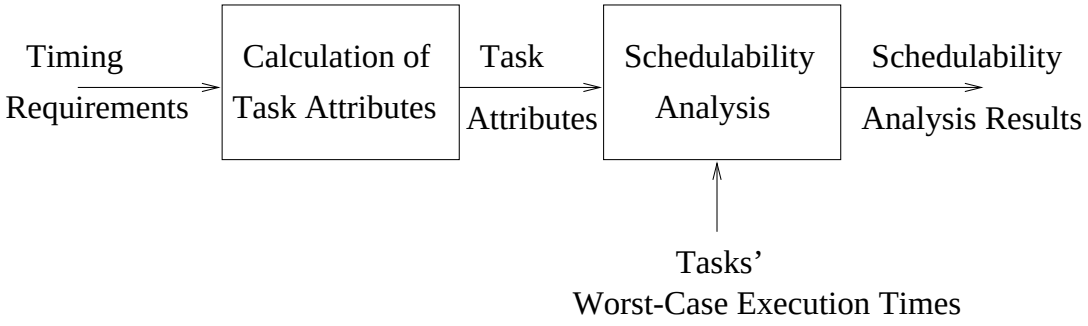


Figure 6.1: Diagram to Illustrate the Approach to Meeting the Timing Requirements

The approach to meeting the timing requirements (and providing evidence that they are met) is illustrated in Figure 6.1. There are basically two parts to this approach; the first involves deriving the task attributes (without knowledge of the tasks' worst-case execution times), and the second then uses the tasks' worst-case execution times to verify that the timing requirements are met.

There are a number of parts to the chapter. Section 6.1 derives analysis for proving transaction requirements are met based on the task set's attributes. Section 6.2 investigates how transaction requirements may be met in a uniprocessor systems in two cases. The two cases are; when the transaction deadline is less than or equal to the tasks' (that form the transaction) periods, and when transaction deadline is greater than the tasks' periods. Section 6.3 investigates how a task's jitter requirement can be met. Section 6.4 investigates how tasks' separation requirements can be met. Finally, section 6.5 combines the product of the earlier sections to provide an overall approach to task attribute assignment.

## 6.1 Calculating the Response Times of Transactions

A prime driver for the task attribute assignment is to represent all the system's timing requirements as task attributes, i.e. if all the tasks' deadline are met then all the system's timing requirements are met. Independent of this aim, it is still useful to have analysis available that allows transactions' requirements to be verified.

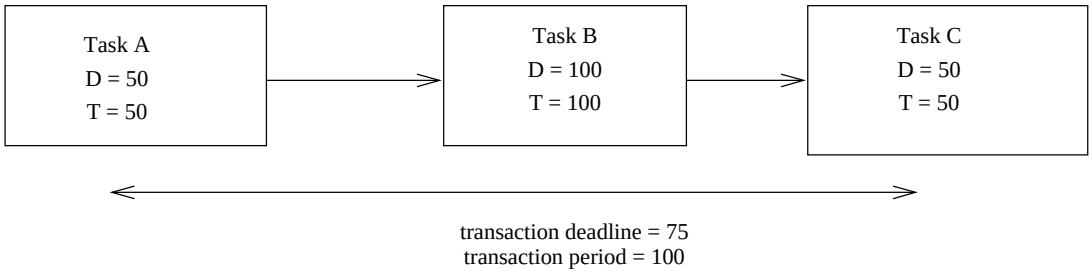


Figure 6.2: Timing Requirements for a Transaction

**Observation 3.** *To verify that transaction requirements are met, it is only necessary to check the one instance of the transaction immediately after the critical instant and only in the case where all tasks execute for their worst-case time.*

**Argument**

Liu and Layland [10] prove that the worst-case response times of tasks occur in the first instance after the critical instant when all tasks execute at their maximum rate and for their maximum execution time. Similarly for a transaction, the worst-case scenario is that the first instance of each task in the transaction is delayed by the maximum amount possible. This causes the maximum amount of interference to the latter tasks in the transaction. Consequently, the final task completes after the longest possible duration from the release of the first task. Therefore, the transaction has its worst-case response. Effectively, this execution scenario for the transaction relates to the critical instant coinciding with each instance of interest (i.e. the one that plays a part in the completion of the transaction) for every task in the transaction.

□

Consider the timing requirements presented in Figure 6.2. Figure 6.2 illustrates a task set consisting of three tasks (A, B and C), and a single transaction requirement across all the tasks. The periods and deadlines of tasks A, B and C are 50, 100 and 50 respectively. The transaction has a period of 100 with a deadline of 75. Unless otherwise specified, all the tasks are initially given a deadline equal to their period. It is common for deadline requirements not to be specified, especially when the requirements are a legacy from a cyclic scheduled system.

There are two phases to the calculation of the transaction's response time, which are: establishing the particular release of each task in the transaction, and the completion time for the releases of interest. This is carried out by starting with the first task and working through the tasks in the defined precedence order. Equation (6.1) can be used for calculating the task instance that is relevant to the transaction.

$$n_t = \left\lceil \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t}{T_t} \right\rceil \tag{6.1}$$

where $t$ is the $t^{th}$ task (assuming tasks are ordered according to precedence) in the transaction,

$n_t$ is the instance of the $t^{th}$ task, where $n_t \in \mathbb{N}$

$n_1$ is 1, and

$R_t^{n_t}$ is the worst-case response time of the $n_t^{th}$ instance of task t.

The worst-case response time of task $t$, $R_t^{n_t}$, of the transaction's critical instance (i.e. time zero) is given in equation (6.2).

$$R_t^{n_t} = (n_t - 1)T_t + R_t \tag{6.2}$$

**Theorem 3.** *The value, $R_t^{n_t}$, represents the worst-case response time of the $t^{th}$ task of the transaction, where the tasks are ordered as defined by the precedence constraint.*

**Proof**

Consider task *t-1*, which is the task in the transaction that precedes the task whose instance is required. The worst-case response time of the $n^{th}$ instance of the task *t-1*, $R_{t-1}^{n_{t-1}}$, can be represented by equation (6.3).

$$R_{t-1}^{n_{t-1}} = (n_{t-1} - 1)T_{t-1} + R_{t-1} \tag{6.3}$$

The next instance of task $t$ must satisfy the condition in equation (6.4), i.e. the response time of task $t$ must be after task $t-1$, but the previous instance of task $t$ must be before task $t-1$.

$$(n_t - 2)T_t + R_t < R_{t-1}^{n_{t-1}} < (n_t - 1)T_t + R_t \tag{6.4}$$

Therefore using equation (6.2),

$$(n_t - 2)T_t + R_t < (n_{t-1} - 1)T_{t-1} + R_{t-1} \tag{6.5}$$

$$n_t < \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + 2T_t}{T_t} \tag{6.6}$$

and,

$$(n_t - 1)T_t + R_t > (n_{t-1} - 1)T_{t-1} + R_{t-1} \tag{6.7}$$

$$n_t > \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t}{T_t} \tag{6.8}$$

We can also state that

$$\forall i \in \text{All Tasks in the Transaction}, n_i \geq 1 \tag{6.9}$$

The conditions in equations (6.6), (6.8) and (6.9) are clearly satisfied by the value of $n_t$ in equation (6.10).

$$n_t = \left\lceil \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t}{T_t} \right\rceil \tag{6.10}$$

□

Using an initial value of $n_1 = 1$, the response time of each task in the transaction can be calculated by starting with the first task and taking each task in turn. Consider the example in Figure 6.2,

$$n_1 = 1 \tag{6.11}$$

$$R_1^1 = R_1 = 50 \tag{6.12}$$

$$n_2 = \left\lceil \frac{R_1 - R_2 + T_2}{T_2} \right\rceil = \left\lceil \frac{50 - 100 + 100}{100} \right\rceil = 1 \tag{6.13}$$

$$R_2^1 = 100 \tag{6.14}$$

$$n_3 = \left\lceil \frac{R_2^1 - R_3 + T_3}{T_3} \right\rceil = \left\lceil \frac{100 - 50 + 50}{50} \right\rceil = 3 \tag{6.15}$$

$$R_3^2 = 150 \tag{6.16}$$

$\therefore$ The worst-case response time for the transaction is 150.

Analysis for verifying transactions can be developed using this approach. However, preference would be given to a scheduling technique that eliminates the need for bespoke analysis other than task schedulability analysis.

## 6.2 Meeting Transaction Deadlines in Uniprocessor Systems

The basic requirement for a transaction is that a sequence of tasks are executed in a specific order within a fixed amount of time. In general, the tasks of the transaction are expected to have identical iteration rates and the transaction deadline is expected to be less than or equal to its period. The reason is tasks of a transaction are dependent on each other and therefore need to be executed at the same rate. However, transactions may exist where the tasks do not execute at the same rate. In this case, it is assumed the transaction repeats at a rate equivalent to the least common multiple of the task periods. Through consultation with systems engineers working on real systems, this assumption is deemed reasonable and acceptable.

Section 2.1 proposes that there are four types of requirements (deadline, jitter, separation and precedence) to be handled by the task attribute assignment. The attributes that can be manipulated are deadline, offset and priority. Some techniques, for example Gerber et al [99], have approached the problem by considering all the attributes of intermediate tasks as changeable. For some systems or some domains, this type of approach is acceptable. However, the problem with the approach is that it neglects highly complex systems where there are many interactions. Changing a task's characteristics may lead to one requirement being met but cause another to be broken. For these reasons, there are two constraints placed on the approach; tasks' periods should not be altered and deadlines can only be reduced. For this reason, the approach developed can be described as unique.

The attributes to be controlled can be further simplified by assigning priorities from deadlines using the DMPO, i.e. the tasks with the shortest deadlines have the highest priority. The requirements could be met just by setting task priorities - in fact the run-time behaviour would be similar. However, the additional analysis given in section 6.1 would be required to prove that the transactions' timing requirements are met. By manipulating task deadlines, the task attributes themselves represent the system's timing requirements. Consequently, the task schedulability analysis also verifies the system's timing requirements.

Another reason for manipulating deadlines instead of priorities is the verification is easier to understand. It is easier to see that the system's timing requirements are met if it can be assumed the task is allowed to execute any time in its possible window [0, $D_i$]. Therefore, simply illustrating the worst-case execution of tasks from the critical instant allows the system's timing requirements to be verified. If the analysis of section 6.1 is used, then just because tasks have a response time less than or equal to their deadline does not mean the system's timing requirements are met.

The proposal is that the necessary timing requirements can be handled by just setting deadlines and offsets to appropriate values. The proposal is demonstrated and proven through the course of this chapter. The system timing requirements are verified by simply proving the task's offsets are enforced in the scheduler and using schedulability analysis to show the task's deadlines are met.

The approach to meeting the transaction requirements is based on reducing the task deadlines in a systematic manner so that the task deadlines are reduced by the minimum possible. The technique is presented through a number of examples building up towards a general-purpose algorithm. An assumption is made that the sum of the worst-case execution times of the tasks is less than or equal to the transaction's deadline.

To demonstrate these approaches work, time-lines are produced as shown in Figure 6.3. The purpose of the time-lines is to illustrate the worst-case response times for tasks and transactions. The time-line depicts the worst-case situation for both tasks and transactions, which is a critical instant for the tasks and the tasks' response times being equal to their deadlines. In accordance with Observation 3, the time-lines present the tasks' execution from the release of the first task of the transaction. The time-lines present enough instances of each task until the required precedence ordering has been achieved, i.e. in the case of the task set in Figure 6.3 task A executes followed by task B followed by task C. It is not deemed necessary to present instances of each task after the instance corresponding to the transaction, e.g. only one release of task A is shown. When producing the time-lines, it is assumed that priorities are calculated using the DMPO.

Each block in the time-lines represents the worst possible response time for the task, which corresponds to the task being released as early as possible and com-

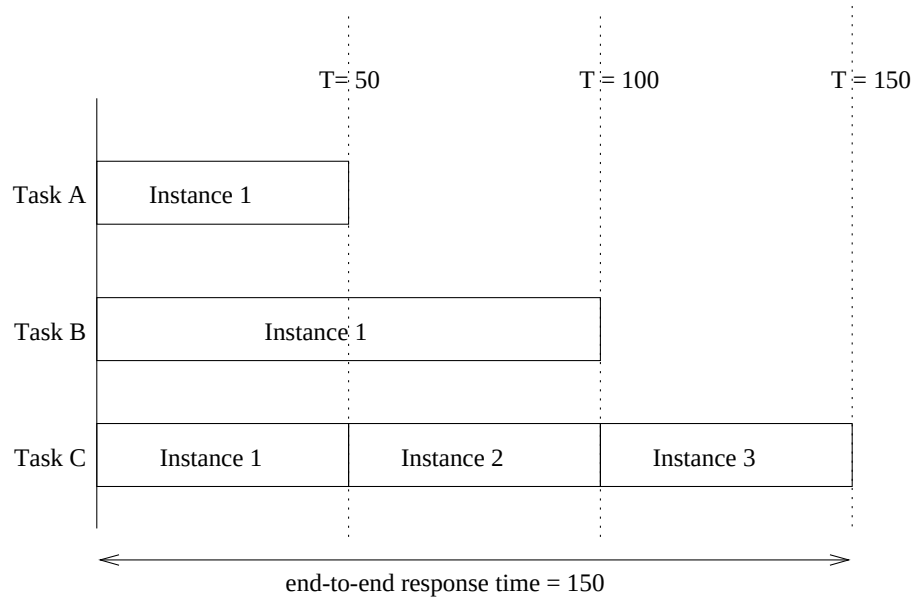


Figure 6.3: A Time-Line for the Transaction Illustrated in Figure 6.2

pleted as late as possible (i.e. at its deadline). The blocks in the time-lines do not represent the actual execution of the tasks. When producing the time-lines, which is effectively the same as analysing whether the transaction requirements are met, the following steps are taken:

1. Assume the first task in the transaction completes at its worst-case time, i.e. at its deadline.

2. Find the appropriate instance of the next (in terms of precedence) task in the transaction. An appropriate instance is one where the response time of the task is greater than the completion time of the preceding task of the transaction.

   *Rule:* If a task has a lower priority than the preceding task and an identical (or later) release time, then the current release can be considered. Otherwise, the next release must be considered.

3. Assume the task completes at its worst-case time, i.e. at its deadline.

4. If the task is not the last of the transaction then return to step 2.

5. Test whether the response time of the transaction meets its deadline.

### 6.2.1 Task Attribute Assignment When the Transaction Deadline $\leq$ Transaction Period

The purpose of this section is to propose a strategy for assigning task attributes so that transaction requirements are met, in the simplest of cases this is when the transactions' deadlines are less than or equal to their periods. The attribute to be controlled is the tasks' deadlines.

An important consideration when trying to meet the transaction's deadline is the relationship between tasks' deadlines. Consider two tasks *t-1* and $t$, task $t$ is always released at the same time as task *t-1* and the tasks have deadlines such that $D_{t-1} < D_t$. Under these conditions, task $t$ always follows task *t-1*, i.e. $n_t = n_{t-1}$ based on equation (6.10). The advantage of the deadlines conforming to this relationship is that the worst-case response time of each task in the transaction is kept to a minimum because the tasks are logically phased. Considering equation (6.2), the response time of the transaction is minimised if the value of $n_i$ (where $i \in$ Tasks in the Transaction) is kept as small as possible. Therefore, the transaction deadline is more likely to be met. In addition, by dealing with deadlines in this way it is easier to justify the precedence ordering is met.

**Observation 4.** *If all the tasks in the transaction are executed in perfect precedence (with the first task of the transaction being first to execute) and the response time of the last task is less than or equal to the transaction's deadline, then the transaction's deadline is met.*

**Argument**

The response time of each task in the transaction is the same as that found through the schedulability analysis equations if a condition is met. The condition is if all the tasks of the transaction execute in perfect precedence with the first task of the transaction being the first to execute. The reason is that for all the tasks ($i$) in the transaction, $n_i = 1$ leading to $R_i^{n_i}$ being equal to $R_i$. Therefore, it is only necessary to check that the worst-case response time of the last task in the transaction is less than or equal to the transaction's deadline.

□

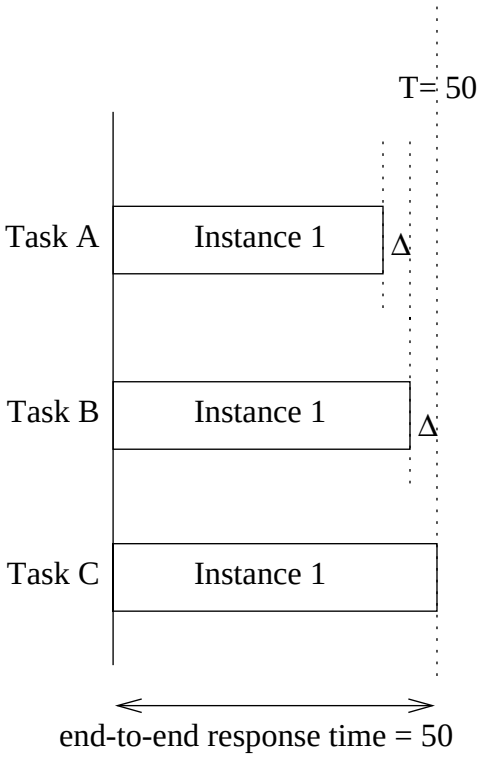


Figure 6.4: A Time-Line for the Transaction in Figure 6.2

To demonstrate the approach to task attribute assignment, consider the timing requirement illustrated in Figure 6.2. The time-line in Figure 6.3 indicates the worst-case end-to-end response time for the task characteristics in Figure 6.2 is 150, which is outside our allowed band.

Tasks' deadlines are assigned starting with the last task in the transaction and working backwards towards the first task. By reducing the deadline of tasks A and B to 50-2$\Delta$ (where $\Delta = 1$ clock cycle) and 50-$\Delta$ respectively, the worst-case response time of the transaction becomes 50 and the requirement is met. The time-line in Figure 6.4 demonstrates the requirement is met.

An important constraint is that deadlines can only ever be decreased because increasing the deadline could affect the ability to meet other requirements. Therefore, if the original deadline of task A was less than $50 - 2\Delta$, then task A's deadline would not be increased to $50 - 2\Delta$. However, the transaction would still be met.

The same run-time effect as that of Figure 6.4 could be obtained simply by altering the priorities of the tasks so that task A has a higher priority than task B and

both have a higher priority than task C. However, the disadvantage of a priority based approach is that extra analysis other than the task schedulability analysis would be required, and yet the resultant priority ordering would be the same. In addition, the constraint (i.e. deadlines) on the tasks are the same because the technique in this section defines the maximum possible set of deadlines. If only tasks' priorities are manipulated, then the tasks still have to meet the deadlines calculated using the approach in this section.

**Observation 5.** *By giving each task in the transaction a deadline at least one clock cycle less than its following task and the final task a deadline less than or equal to the transaction's deadline, the transaction's requirement are met if the tasks are schedulable.*

**Argument**

If DMPO is used then, if the preceding task in the transaction has a larger deadline than the next task being considered, then its deadline must be made less in order to maintain the precedence ordering. Using DMPO, the deadline only needs to be made one clock cycle less than the following task. With a perfect precedence order, for the transaction to meet its deadline the last task of the transaction must complete by the transaction's deadline. Therefore, if the deadline of the last task is initially greater than the transaction deadline, then it has to be altered so it is equal to the transaction deadline. Hence, the task attributes also represents the transaction requirements, leading to the schedulability analysis verifying the transaction requirements as well as the task requirements.

□

In fact, Observations 4 and 5 can be combined to give Observation 6

**Observation 6.** *If the transaction's deadline is less than or equal to the transaction's period, then reducing tasks' deadlines by the minimum necessary (as advocated by the technique in this section) to achieve perfect precedence is optimal. A definition for optimal in this domain can be taken as an approach finds a solution when one exists.*

**Argument**

Observation 5 argues that the transaction requirement is met if the priority ordering discussed in this section is used. From Observation 5, it can also be stated that the maximum deadlines for the tasks result and the minimum necessary changes are made to the deadlines to achieve perfect precedence. Therefore based on the fact that DMPO is optimal [10], the method of assigning task attributes to cater for transaction requirements is optimal.

□

It has been argued that the task assignment algorithm in this section is optimal in the case where the transaction's deadline is less than its period. However in our experience, the computational model considered in this section is too restrictive because transaction deadlines is often greater than their periods. The following section is to relax this restriction.

### 6.2.2 Task Attribute Assignment For Arbitrary Transaction Deadlines

The aim of this section is to develop an approach to task attribute assignment for cases where transactions deadlines have an arbitrary value. Again, the aim is to use deadline assignment instead of priority assignment to avoid the need to separately analyse whether transaction requirements are met.

A transaction's deadline could be dealt with using the approach in section 6.2.1. However, the ability to schedule the task set may be unnecessarily reduced by making the deadlines less than needed. The reason is the transaction's deadline would be constrained so it has to be less than or equal to its period. A potential solution to this problem is to increase the deadline of the last task in the transaction to the value of the deadline of the transaction. Then, the task deadlines would be assigned so that precedence is achieved as described in section 6.2.1. The transaction's deadlines would then be verified using the schedulability analysis for arbitrary deadlines discussed in section 3.1. There are two problems with adopting this approach, which are; this form of timing analysis is pessimistic [70], and the tasks may have other constraints, e.g. other transactions, preventing the deadlines being increased. Therefore, a more appropriate approach is sought.

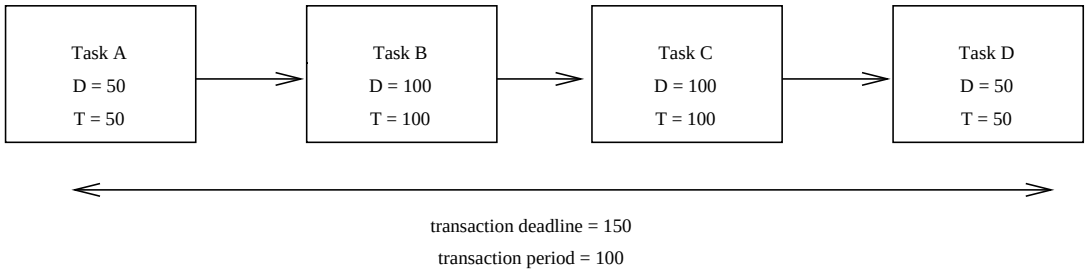


Figure 6.5: Timing Requirements for a Transaction

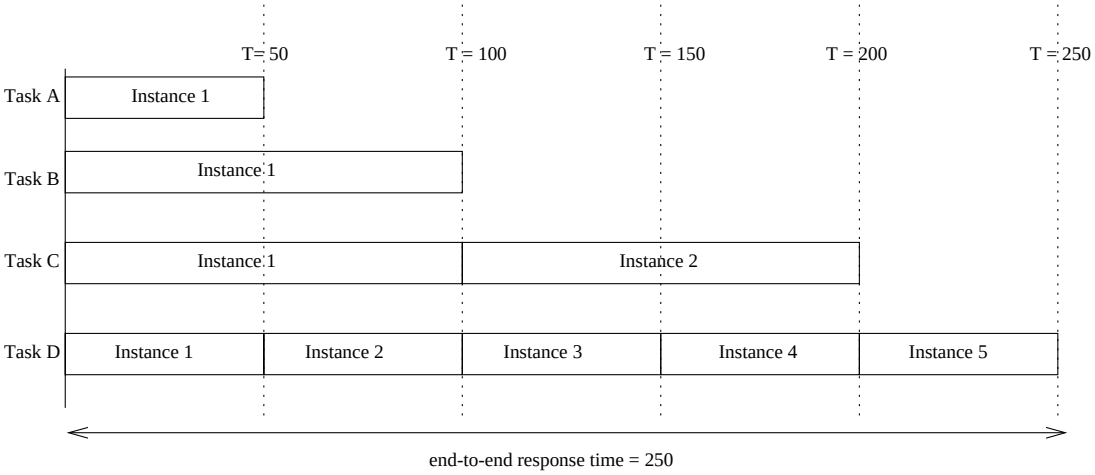


Figure 6.6: A Time Line for the Transaction in Figure 6.5

A characteristic of the requirements is a task's period could be longer than the period of the task that follows it in the transaction. In this case, it may not be necessary or possible to reduce all the deadlines so that perfect precedence between the tasks of the transaction are maintained. For example, consider the timing requirements in Figure 6.5. The requirements of the transaction are an iteration rate of 100 and an end-to-end deadline of 150. The periods of tasks A, B, C and D are 50, 100, 100 and 50 respectively. In this case, task B cannot always follow task A because task B has a slower update rate.

Figure 6.6 shows that without altering tasks' deadlines, the requirement is not met because the transaction's response time is 250. By the previous approach, in section 6.2.1, the transaction requirements could be met as shown in Figure 6.7. The task attributes illustrated in Figure 6.7 are sub-optimal because the timing requirements could be met with larger deadlines as shown in Figure 6.8. Having larger deadlines means the tasks have a lower priority. The benefit of this is that

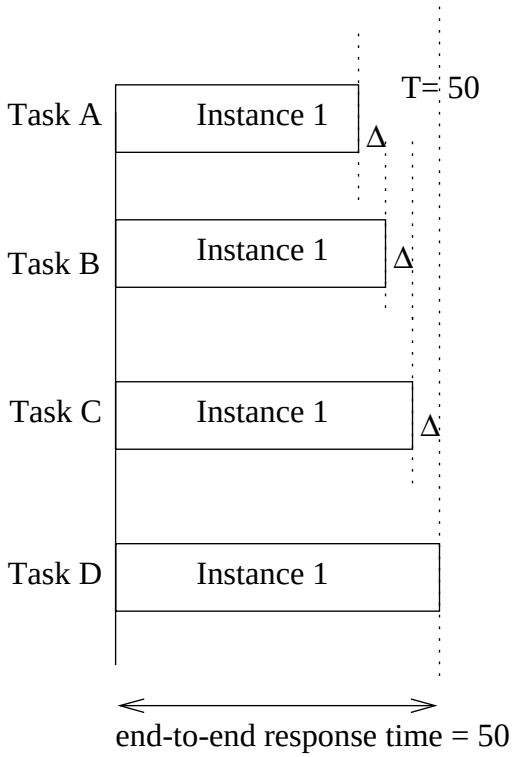


Figure 6.7: A Time Line for the Transaction in Figure 6.5

the impact of the change in deadlines on the rest of the system is minimised and the tasks are more likely to be schedulable.

The task attributes of Figure 6.7 can be generated using Algorithm 1. The algorithm does not simply rely on the tasks' deadlines being reduced so that perfect precedence is maintained. Instead, while the transaction requirement is not met, the highest deadline is reduced by $\Delta$, and if this causes any of the preceding sequence of tasks in the transaction to have the same deadline, then its deadline is also reduced by $\Delta$. In the case of the task set in Figure 6.5, tasks B and C have their deadlines repeatedly reduced until the transaction's deadline requirement is met.

The benefit of the approach in Algorithm 1 is that if the deadlines are larger, then there is a greater chance of a schedulable solution. The algorithm also contains a manipulation of the *task_deadlines_are_changing* flag. The use of this flag becomes apparent later in the chapter.

**Algorithm 1.** - Algorithm for the Generalised Approach

*for each task in the transaction*
  *if the following task has an equivalent deadline then*
    *reduce the task's deadline by* $\Delta$
    *assign the value of false to the task_deadlines_are_changing flag*
  *while the transaction deadline is not met*
    *assign the value of true to the task_deadlines_are_changing flag*
    *take the longest deadline and reduce it by* $\Delta$
    *for each task in the transaction*
      *if the following task has an equivalent deadline then*
        *reduce the task's deadline by* $\Delta$
        *assign the value of false to the task_deadlines_are_changing flag*

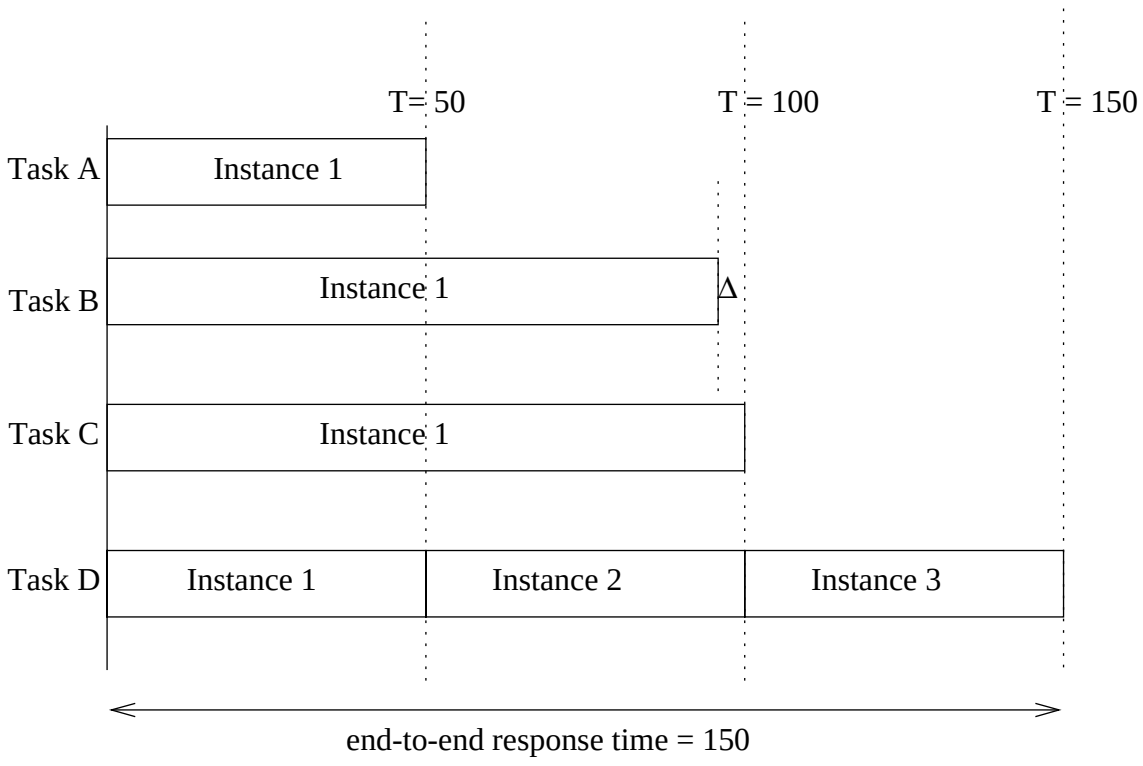


Figure 6.8: A Time Line For the Transaction in Figure 6.5

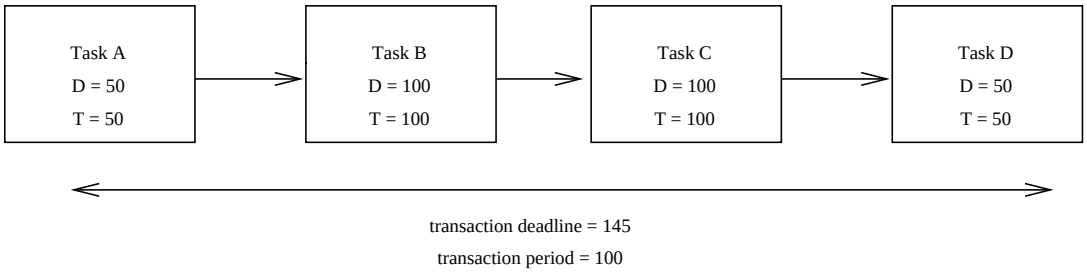


Figure 6.9: Timing Requirements For a Transaction

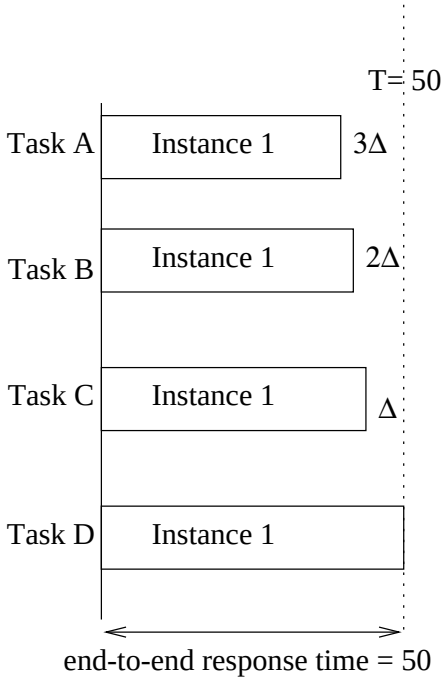


Figure 6.10: A Time Line for the Transaction in Figure 6.9

### 6.2.3 Optimality of the Approach

With any approach it is important to ascertain whether it is optimal. If it is not optimal, then any deficiencies should be highlighted so that appropriate action can be taken where necessary. First impressions suggest that the approach may be optimal because the deadlines of the tasks are the maximum possible whilst still meeting the timing requirements. However, trying to prove optimality is difficult because of the effect of complex interactions with other tasks and transactions.

Consider the transaction requirement in Figure 6.9. Figure 6.10 presents the solution derived using the approach in Algorithm 1, whereas Figure 6.11 presents

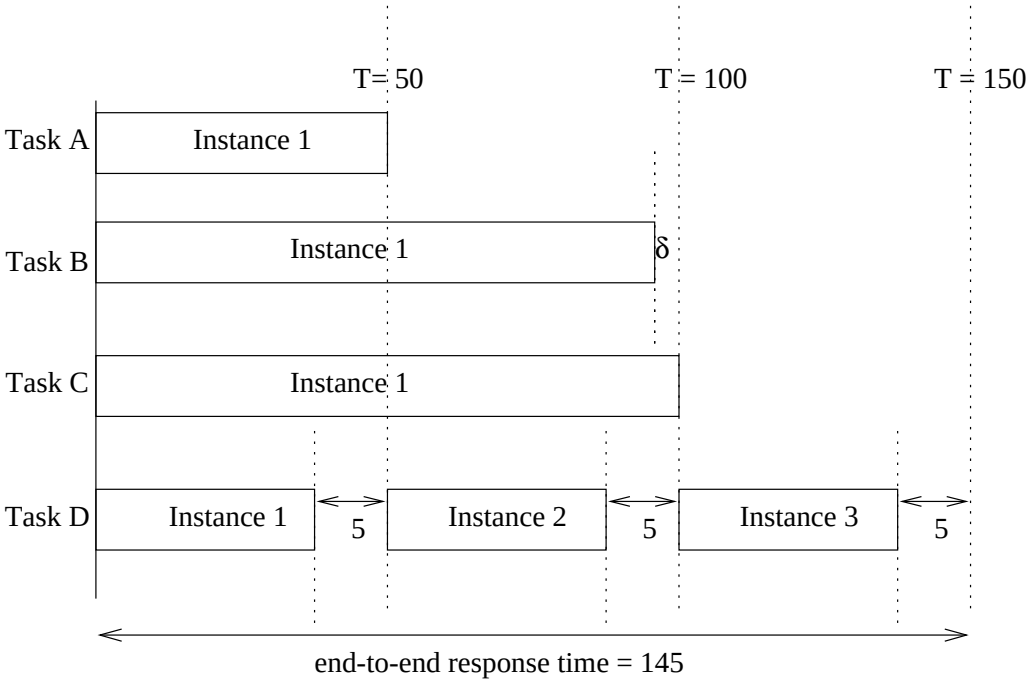


Figure 6.11: A Time Line for the Transaction in Figure 6.9

an alternative solution. It is impossible to judge which of these is the better solution without knowledge of all the tasks' execution times, priorities and deadlines. On the one hand, the task attributes in Figure 6.10 cause more interference through tasks B and C. While in Figure 6.11, tasks B and C cause less interference but the interference associated with task D is increased. Therefore, it can be stated that the approach in Algorithm 1 is not optimal.

In our experience using the task attribute assignment approach, no specific examples have been found where the issue raised in this section has caused problems. Chapter 7 contains an example of how the technique is used to develop tasks' attributes for a complex set of "real" system timing requirements. Where the approach in Algorithm 1 is insufficient, then a solution could be based on the optimal priority assignment technique given by Audsley [54]. Audsley's approach involves methodically raising the priority of tasks that are unschedulable, until the task set is schedulable or a solution is not deemed possible. However, Audsley's technique is not intended to deal with transaction requirements. Therefore, Audsley's approach would have to be extended to check all the system's timing requirements before making a decision whether to change priority. The advantages of the approach in this section over an approach based on Audsley's optimal

priority assignment algorithm are: the use of time-lines helps convince the engineer of the approach's correctness, and the approach has significantly lower computational complexity.

## 6.3 Jitter

The purpose of this section is to explore how jitter requirements can be handled by the manipulation of task attributes. An important constraint, and a frequent criticism of fixed priority scheduling is related to the control of jitter. A conventional model of fixed priority scheduling is said to suffer worse jitter than a cyclic scheduler [6]. The reason is the regular clock tick in the cyclic scheduler causes the entire run queue to be regularly refreshed. Therefore, a task suffers less jitter than the length of one minor cycle. Whereas with the fixed priority scheduler, tasks with large periods and deadlines can exist on the run queue for a longer time, leading to greater variations in their completion time. Section 2.1 discusses how jitter requirements are important for restricting the variability in the time when inputs and outputs are performed.

A prime driver for this work is to enable the schedulability analysis for tasks to also verify the jitter requirements. Therefore for similar reasons to those given in section 6.2.1, the jitter requirement should be handled using deadlines rather than priorities. If a task has a deadline equal to the jitter requirement, then the requirement is met because the window of allowed execution is constrained between the critical instance and the time allowed for the jitter requirement. Using this approach would be pessimistic since there is a minimum processing time for each task that adds no variability, i.e. the task's execution time can only vary between its best-case response time and its worst-case response time. Therefore, the deadline can be calculated using equation (6.17).

$$D_i = J_i + BCRT_i \tag{6.17}$$

where $BCRT_i$ is the best case response time of task i.

The best case response times of tasks can be generated in two stages. Initially, the value can be set to zero, this makes the task set harder to schedule since the deadlines are lower. Later, the value can be evolved as more results from

analysis become available. The best case response times can be taken as the best case execution time or calculated using either exact analysis techniques, or the method proposed by Harbour, Garcia and Guiterrez [100].

Clearly it can be seen that having a deadline as defined in equation (6.17) constrains the variation of the task's computation time so that the jitter requirement is met. The fact that the choice of deadline controls the jitter means that the standard schedulability test also verifies the jitter requirement is met removing the need for extra analysis. However, there are problems with the approach based on equation (6.17). Two cases are illustrated.

**Case 1**

Whilst the technique described in this section provides task attributes that result in the requirements being met, it is not optimal. The main problem is that if many tasks have jitter requirements, then there may be an abnormally large number of tasks with short deadlines. Short deadlines means the tasks have to execute shortly after the critical instant, which could easily lead to an unschedulable solution.

**Solution for Case 1**

Phase the execution of tasks by spreading the execution using offsets. However, the use of offsets to phase execution should be avoided, where possible, due to the increased maintenance problems of having "slots". The maintenance problems of slots would be analogous to the problems of cyclic scheduling.

□

**Case 2**

Another area of concern is related to transactions, where the transaction deadline is less than or equal to its period. A realistic assumption is that any jitter requirement imposed on the transaction invariably relates to the first and last task in the transaction, since these tasks are the input and output of the transaction. A jitter requirement placed on the last task in the transaction would lead to all the preceding tasks having a shorter deadline, which could affect schedulability. For example, consider the task set illustrated in Figure 6.12, which represents a transaction requirement that includes a jitter requirement on the last task.

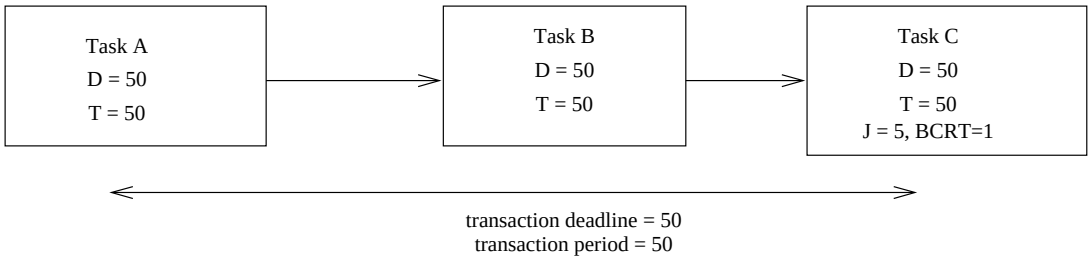


Figure 6.12: Timing Requirements For A Transaction

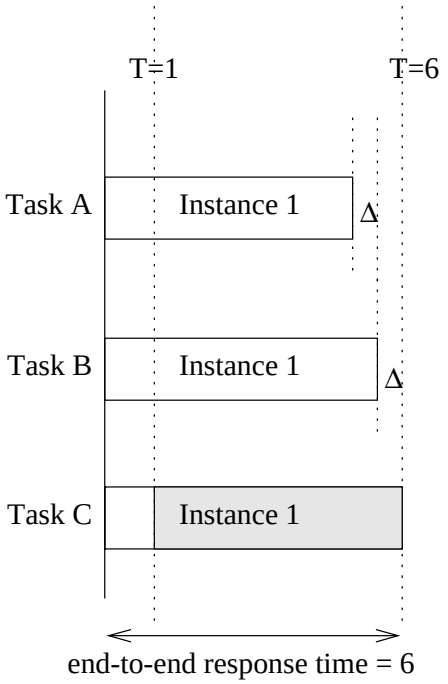


Figure 6.13: Time-line For the Transaction in Figure 6.12

Using equation (6.17) to calculate the jitter requirement of task C results in a deadline of 6. Then applying the technique in Algorithm 1 leads to tasks A and B having a deadline of $6-2\Delta$ and $6-\Delta$. The time-line in Figure 6.13 illustrates the execution of the tasks. The shaded area represents the allowed variation in task C's execution and the other time (1 unit) is the non-variant (equal to task C's best case response time) part of its execution. The problem with the solution in Figure 6.13 is that all the tasks have to execute within a tight deadline.

### Solution for Case 2

The solution to the problem is again the use of offsets. It is proposed that the solution demonstrated in Figure 6.14 is often better than the solution demon-

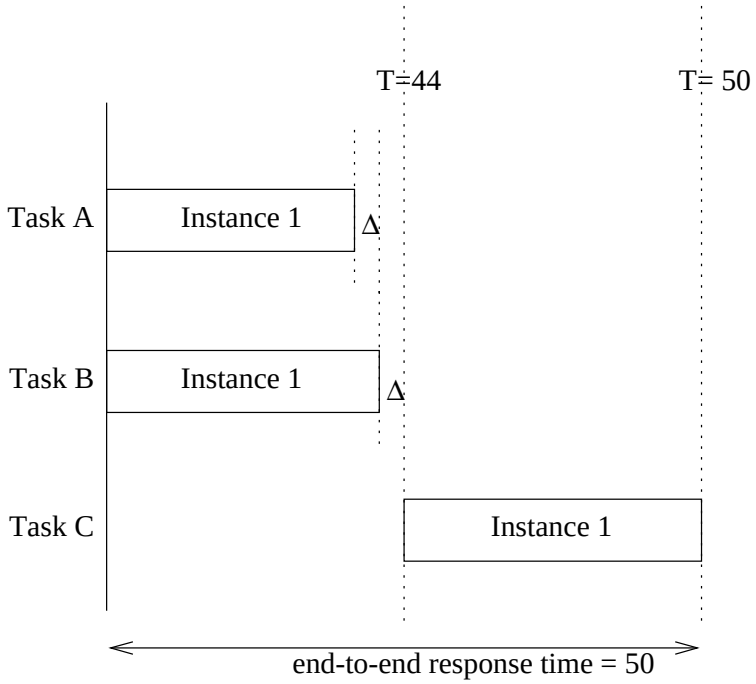


Figure 6.14: Time Line for the Transaction in Figure 6.12

strated in Figure 6.13. The basis of the solution is to constrain the variation in task C's execution at the end of the allowed execution time of the transaction by using an appropriate offset. The offset is calculated as shown in equation (6.18). The allowed execution time for task C is equal to the deadline calculated using equation (6.17). Precedence of the other tasks is enforced by making the other tasks execute before task C is released. Therefore, task B is given a deadline equal to the offset of task C and then Algorithm 1 is applied to the tasks preceding task C. The approach for choosing task B's deadline and task C's offset can be represented by Algorithm 2.

$$\begin{aligned} \text{offset} &= \text{transaction deadline} - \\ &\quad \text{time allowed in the execution window for task C} \\ \text{offset} &= \text{transaction deadline} - (J_C + BCRT_C) \end{aligned} \tag{6.18}$$

□

It should be noted that if the transaction's deadline is greater than its period, then the problem in Case 2 does not arise because Algorithm 1 does not unnecessarily enforce precedence in this case. Instead, the longest deadline is reduced

**Algorithm 2.** - Algorithm for Dealing With Jitter

*for each task (denoted i) in the system*
  *if task i has a jitter requirement then*
    *if task i is not the last task in the particular transaction then*
      $D_i = J_i + BCRT_i$
    *else*
      *if the transaction deadline > transaction period then*
        $D_i = J_i + BCRT_i$
      *else*
        *if* $D_i$ *> transaction deadline then*
          *task deadline = transaction deadline*
        $O_i = D_i - (J_i + BCRT_i)$
        *for all preceding tasks (denoted j) in the transaction*
          *if transaction deadline is not met then*
            *if* $D_j$ *> transaction deadline - (*$J_i + BCRT_i$*) then*
              $D_j$ *= transaction deadline - (*$J_i + BCRT_i$*)*

until the requirements are met. Therefore, reducing task C's deadline to meet its jitter requirement does not necessarily lead to tasks A and B's deadlines being reduced to enforce the precedence.

The approach defined in Algorithm 2 cannot be described as optimal. If a large number of tasks (specifically ones that are not part of the particular transaction) are already phased to execute during the time interval $[O_i, D_i]$, then the approach may lead to an unschedulable solution. In this circumstance an approach based entirely on equation (6.17) and Algorithm 1 may increase the likelihood of a schedulable solution. However, in practice this is unlikely to be the case.

There is no optimal approach published for dealing with jitter.

## 6.4 Separation

The purpose of this section is to present a technique for assigning task attributes so that separation requirements are handled. For two tasks with a separation

**Algorithm 3.** - Algorithm that Accounts for Separation

*for each task in the system*
  *if the task (denoted i) is to be separated from an earlier task (denoted i-1) then*
    $O_i = S_i + D_{i-1}$
  *if* $(O_i + C_i) > D_i$ *then*
    $D_{i-1} = \frac{D_i - S_i - O_{i-1}}{2}$
    *assign the value of true to the task_deadlines_are_changing flag*

requirement ($S_i$) between them, the requirement can be satisfied by manipulating the offsets and deadlines of the two tasks. The technique gives the second of the two tasks an offset $S_i$ from the deadline of the earlier task. This is expressed in equation (6.19).

$$O_i = S_i + D_{i-1} \tag{6.19}$$

where $S_i$ is the separation requirement between two tasks,
task *i-1* precedes task $i$, and
$D_{i-1}$ is the deadline of task *i-1*.

If the resultant offset is too great (i.e. $O_i + C_i > D_i$), then task $i$ does not have a chance to meet its deadline. In this case, an appropriate metric for altering the deadline of the first task is required. An approach is presented in equation (6.20). The approach basically splits the available execution time between the two tasks. This is achieved by making the relative deadline (equal to the task's deadline minus its offset) of the preceding task equal to the relative deadline of the current task. The condition in equation (6.19) still has to be satisfied. The approach developed is represented in Algorithm 3. However, it should be noted that manual intervention may still be deemed necessary, dependence on the characteristics of the system.

$$D_{i-1} = \frac{D_i - S_i - O_{i-1}}{2} \tag{6.20}$$

**Algorithm 4.** - Algorithm for Priority Assignment

*initialise task_deadlines_are_changing flag to true*
*apply algorithm 2*
*while the task_deadlines_are_changing flag is true*
    *assign the value of false to the task_deadlines_are_changing flag*
    *for each transaction in the set of transactions*
        *apply algorithm 1*
    *apply algorithm 3*
*apply priorities to the tasks by the deadline monotonic approach*
*perform schedulability analysis of the task set*

## 6.5 Overall Task Attribute Assignment

Combining the various techniques discussed in this chapter results in Algorithm 4. The algorithm works in the following steps:

1. If a task does not already have a deadline, then it is assigned a deadline equal to its period.
2. Tasks with a jitter requirement are assigned a deadline using Algorithm 2.
3. For each transaction in the system use Algorithm 1 to update task deadlines to meet the transaction requirement.
4. Apply Algorithm 3 to deal with separation requirements.
5. If a complete run through all the transactions causes a change of deadline to take place (i.e. *task_deadlines_are_changing flag* is true), then return to step 3.
6. Assign priorities using DMPO.
7. Check that each task's deadline is met using the schedulability analysis.

The overall method for priority assignment is not considered to be optimal because of some particularly obscure events that are discussed in this chapter. For

example, the case in section 6.2.3. However, use has shown that the approach is sufficient for the typical system timing requirements encountered as demonstrated by the case study in Chapter 7.

The only real problem encountered so far when using this approach has been caused by impractical requirements related to circular arguments. For instance consider the following two requirements, one requirement is task A has to precede task B and the other requirement is task B has to precede task A. Both requirements can not be resolved by Algorithm 1. In this case the algorithm 1 would never obtain a result, with the deadlines tending towards $-\infty$. The reason is that if transactions' deadlines are less than their periods, then the requirement can only be met if perfect precedence is achieved. To achieve perfect precedence, the earlier task in the transaction would have its deadline reduced as shown in Algorithm 1. This process is repeated for both of the transactions until the tasks' attributes converge, which will not happen. Clearly, there is no solution to this requirement, therefore it can be stated the requirement should not be allowed to exist. When implementing the algorithm, circularities should be recognised and flagged to the user so that appropriate manual intervention can take place.

The ability of Algorithm 4 to deal with complex requirements is further demonstrated with the case study in Chapter 7. The case study is used to generate the priorities for a complex set of realistic timing requirements.

The following is a comparison of the task attribute assignment approach in this chapter with the four chosen criteria for successful technology transfer:

1. *Reuse* - The reuse criterion is not really applicable because any technique derived for fixed priority scheduling cannot be expected to work for the cyclic scheduler. However, at least the approach means that only a single tool is needed for analysis.

2. *Certification* - The technique provides an approach where it can be determined by drawing time-lines of the worst-case execution from the critical instant that the system's timing requirements are met. Therefore, evidence of correctness can be expressed in a manageable form.

3. *Sufficiency* - The approach places the minimum constraints possible on the execution of tasks and hence it provides the greatest flexibility. Therefore,

the likelihood of being able to schedule the system is improved over a less flexible approach.

4. *Understanding* - In our experience, engineers (including the certification authorities) have been able to understand and apply the technique in a short time (typically less than an hour) even with little previous knowledge of scheduling. The satisfaction of the criterion is demonstrated by the fact the approach has been technology transferred [96] to industry, where it has been successfully applied .

## 6.6 Contrast With Other Techniques

One of the other benefits of this priority assignment approach is that the need for any form of priority inheritance is eliminated. The need for the priority inheritance protocols is eliminated because the sharing of resources can be dealt with at the priority level. Priority inheritance was borne out of the inflexibility of the rate monotonic approach to deal with transactions in any other way.

Consider the task set in Figure 6.2 with task attributes assigned using the rate monotonic approach, the response time of the system is the same as shown in Figure 6.3 where the deadline is missed. To meet the transaction's deadline when there is no control over tasks' deadlines requires tasks A and B to inherit a higher priority from task C. There are a number of mechanisms for inheriting priorities reviewed in section 3.1.7. The priority inheritance would give a similar response to the one shown in Figure 6.4. The problem with priority inheritance is that it makes the design more complex and there are additional run-time overheads. Another problem is that extra verification, other than the schedulability analysis, is required to show the transaction's requirements are met.

Another technique that the approach can be compared with is that of Gerber [99] discussed in section 3.1.7. There are many differences between the techniques, but two main ones exist. Firstly, with Gerber's approach it is not straightforward to demonstrate how the requirements are met. Secondly, Gerber's approach assumes that intermediate task attributes are completely flexible (i.e. iteration rates can be changed and deadlines increased) - this ignores the influence of other timing requirements, e.g. the interaction with other tasks and transactions.

Therefore for the domain of interest, the technique developed in this chapter has many advantages.

## 6.7 Summary

This chapter has addressed how attributes can be assigned to tasks that represent the system's timing requirements as priorities, offsets and deadlines. The three primary benefits of this work are:

1. the standard schedulability analysis that assumes a critical instant can verify all the system's timing requirements,
2. the approach is easy to understand, and
3. the approach removes the need for complex mechanisms such as priority inheritance.

The approach derived has the significant advantage that enforcing tasks' precedence rules is dealt with off-line, rather than the kernel having to enforce the precedence rules at run-time. This makes the implementation easier to produce and maintain, and it is easier to verify the precedence constraints are achieved. In most cases the approach is sufficient, except for a few obscure cases. An additional advantage is that our approach can be explained relatively straight forwardly to engineers and regulators, which helps the technology transfer process. The ease in which time-lines can be produced assists technology transfer and helps when arguing correctness.

The techniques described in this section are illustrated in Chapter 7 as part of a case study. The purpose of the case study is not only to demonstrate the techniques further, but to show how they may be applied to a real system.

# Chapter 7

# Case Study - The BR715 Engine Controller

The purpose of this chapter is to present a case study that demonstrates how the techniques described in Chapters 4 - 6 could be used for a *real* system. The chapter is to demonstrate the techniques are fit for purpose, particularly in relation to the four criteria for successful technology transfer. The criteria are: reuse, certification, understanding, and sufficiency.

The case study is based on work performed for Rolls-Royce that moved an aircraft electronic engine controller application from using cyclic scheduling to fixed priority scheduling [36, 96, 101, 102, 103]. The technical work was originally performed by the author, however the work has since been adopted by Rolls-Royce. The techniques have lead to a system scheduled with fixed priority scheduling to be used and verified on the actual engine. Discussions have taken place with the certification authorities about the use of the fixed priority scheduling technique on a future aircraft - its use has been agreed [96].

For confidentiality reasons some of the details presented have been changed, e.g task names. However, the example is still realistic. The discussion is divided into a number of parts:

1. The actual purpose of the electronic engine controller,
2. Technical details concerning the system as controlled by the cyclic scheduler,

3. Technical details of how the change to fixed priority scheduling was performed, including the timing analysis, and

4. Process details of how the change to a fixed priority scheduler affect the working practice within the organisation.

## 7.1 Purpose of the Electronic Engine Controller System

Modern aircraft engines are fitted with an electronic engine controller, which is essentially a computer that executes engine controller software. Before the use of computers on aircraft, the control functionality would have been implemented using hydro-mechanical components.

An overview of the operation of the system is given in Figure 7.1. The electronic engine controller uses sensors to monitor the engine condition (e.g. fuel flow) and other components to monitor the aircraft operation (e.g. thrust request). The electronic engine controller controls the engine via the operation of actuators, such as valves, ignitors and pumps. The electronic engine controller also accepts pilot commands, and provides status information about the engine back to the cockpit. The electronic engine controller is embedded and safety-critical, normally featuring replicated components to provide fault tolerance.

A particular characteristic of electronic engine controllers is that transactions are common place and fundamental to safe operation. The transactions consist of reading data from a number of sensors, performing calculations based on the available data and the output of results to the appropriate actuators. The system also has tasks that provide functionality such as health monitoring and maintenance. Other important factors are the implemented system is normally uniprocessor and the processor's resource is often heavily loaded.

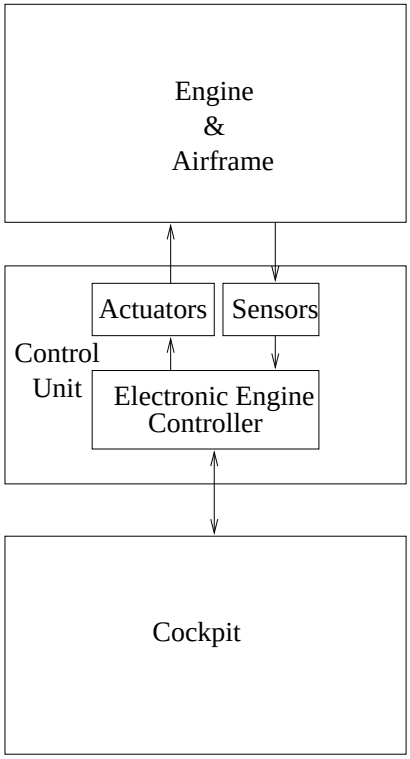


Figure 7.1: Overview of an Electronic Engine Control Unit

## 7.2 Technical Details of the System

The purpose of this section is to present enough details of how the system is implemented using the cyclic scheduler in order to place the transition in context. The section uses the discussion in Chapter 2 of the existing ways of implementing a cyclic scheduler as a basis.

The basic infrastructure of the system is a periodic clock tick (period = 25000 - the time units are arbitrary). The clock tick is used to signify the start of a minor cycle and to control the timing watchdog (implemented in hardware). The implementation features eight minor cycles per major cycle. The overhead is:

$$C_{TW} = 100$$

where $C_{TW}$ is the worst-case execution time of the timing watchdog software.

The timing requirements and certain characteristics of the system are presented in Figure 7.2 and Table 7.2. Figure 7.2 presents the transaction details for the

system, where the arrows represent the precedence ordering (i.e. left to right in the diagram) between tasks and the transaction deadline is associated with the last task (i.e. the right-most) and is contained in boxes surrounded by a dashed line. It should be noted that only the tasks that are part of the transactions are shown in Figure 7.2. The transaction deadlines are:

1. All transactions that end with task $P11$ have a deadline of 75000. The exception is the transaction that features tasks $P15$, $P3$ and $P11$, where the transaction deadline is 50000.

2. All transactions that end with task $P35$ have a deadline of 150000.

3. All transactions that end with task $P21$ have a deadline of 75000.

The task details (identifier, jitter, and period) are contained in boxes surrounded by a solid line. For example, task $P11$ has a period of 25000, and a jitter requirement of 12500. Where a jitter is not stated, no jitter requirement exists.

Table 7.1 presents the timing characteristics for the tasks and has five columns. The columns fall into two categories: timing requirements and task attributes. The columns are:

**Timing Requirements**

$Id$ is the identifier
$T$ is the period
$D$ is the deadline - normally equal to the period
$J$ is the jitter

**Task Attribute**

$C$ is the worst-case execution time

From Figure 7.2 and Table 7.1, it can be seen that the timing requirements are not trivial. The processor utilisation is 85.28% without any overheads being taken into account. This represents a tight timing margin, which makes the synthesis of task attributes difficult. The timing margin also makes the system difficult to maintain. A large benefit of moving to a fixed priority scheduler is

| ID | T | C | D | J |
|---|---|---|---|---|
| P21 | 25000 | 684 | 25000 | 12500 |
| P35 | 50000 | 173 | 50000 | 12500 |
| P3 | 25000 | 461 | 25000 | 12500 |
| P1 | 25000 | 300 | 25000 | 0 |
| P2 | 25000 | 2088 | 25000 | 0 |
| P4 | 25000 | 340 | 25000 | 0 |
| P5 | 25000 | 7 | 25000 | 0 |
| P6 | 25000 | 85 | 25000 | 0 |
| P7 | 25000 | 1910 | 25000 | 0 |
| P8 | 25000 | 1971 | 25000 | 0 |
| P9 | 25000 | 640 | 25000 | 0 |
| P10 | 25000 | 17 | 25000 | 0 |
| P12 | 25000 | 103 | 25000 | 0 |
| P13 | 25000 | 203 | 25000 | 0 |
| P14 | 25000 | 26 | 25000 | 0 |
| P15 | 25000 | 14 | 25000 | 0 |
| P16 | 25000 | 408 | 25000 | 0 |
| P17 | 25000 | 278 | 25000 | 0 |
| P18 | 25000 | 190 | 25000 | 0 |
| P19 | 25000 | 32 | 25000 | 0 |
| P20 | 25000 | 228 | 25000 | 0 |
| P22 | 25000 | 273 | 25000 | 0 |
| P23 | 25000 | 1265 | 25000 | 0 |
| P24 | 50000 | 318 | 50000 | 0 |
| P25 | 100000 | 1334 | 100000 | 0 |
| P26 | 50000 | 52 | 50000 | 0 |
| P27 | 200000 | 796 | 200000 | 0 |
| P28 | 50000 | 336 | 50000 | 0 |
| P29 | 50000 | 408 | 50000 | 0 |
| P30 | 50000 | 798 | 50000 | 0 |
| P31 | 100000 | 457 | 100000 | 0 |
| P32 | 50000 | 351 | 50000 | 0 |
| P33 | 50000 | 390 | 50000 | 0 |
| P34 | 50000 | 201 | 50000 | 0 |
| P36 | 50000 | 925 | 50000 | 0 |
| P37 | 50000 | 321 | 50000 | 0 |
| P38 | 50000 | 1801 | 50000 | 0 |
| P39 | 50000 | 522 | 50000 | 0 |

| ID | T | C | D | J |
|---|---|---|---|---|
| P40 | 50000 | 256 | 50000 | 0 |
| P41 | 100000 | 196 | 100000 | 0 |
| P42 | 50000 | 900 | 50000 | 0 |
| P43 | 50000 | 1945 | 50000 | 0 |
| P44 | 100000 | 528 | 100000 | 0 |
| P45 | 100000 | 551 | 100000 | 0 |
| P46 | 100000 | 272 | 100000 | 0 |
| P47 | 100000 | 271 | 100000 | 0 |
| P48 | 100000 | 378 | 100000 | 0 |
| P49 | 100000 | 107 | 100000 | 0 |
| P50 | 100000 | 217 | 100000 | 0 |
| P51 | 100000 | 4698 | 100000 | 0 |
| P52 | 100000 | 232 | 100000 | 0 |
| P53 | 100000 | 30 | 100000 | 0 |
| P54 | 100000 | 763 | 100000 | 0 |
| P55 | 100000 | 62 | 200000 | 0 |
| P56 | 200000 | 304 | 200000 | 0 |
| P57 | 200000 | 336 | 200000 | 0 |
| P58 | 200000 | 100 | 200000 | 0 |
| P59 | 200000 | 8 | 200000 | 0 |
| P60 | 200000 | 378 | 200000 | 0 |
| P61 | 200000 | 38 | 200000 | 0 |
| P62 | 200000 | 428 | 200000 | 0 |
| P63 | 200000 | 2258 | 200000 | 0 |
| P64 | 200000 | 328 | 200000 | 0 |
| P65 | 1000000 | 5040 | 1000000 | 0 |
| P66 | 1000000 | 5040 | 1000000 | 0 |
| P67 | 1000000 | 5040 | 1000000 | 0 |
| P68 | 1000000 | 5040 | 1000000 | 0 |
| P69 | 1000000 | 5040 | 1000000 | 0 |
| P70 | 1000000 | 5040 | 1000000 | 0 |
| P71 | 1000000 | 5040 | 1000000 | 0 |

Table 7.1: Task Attributes

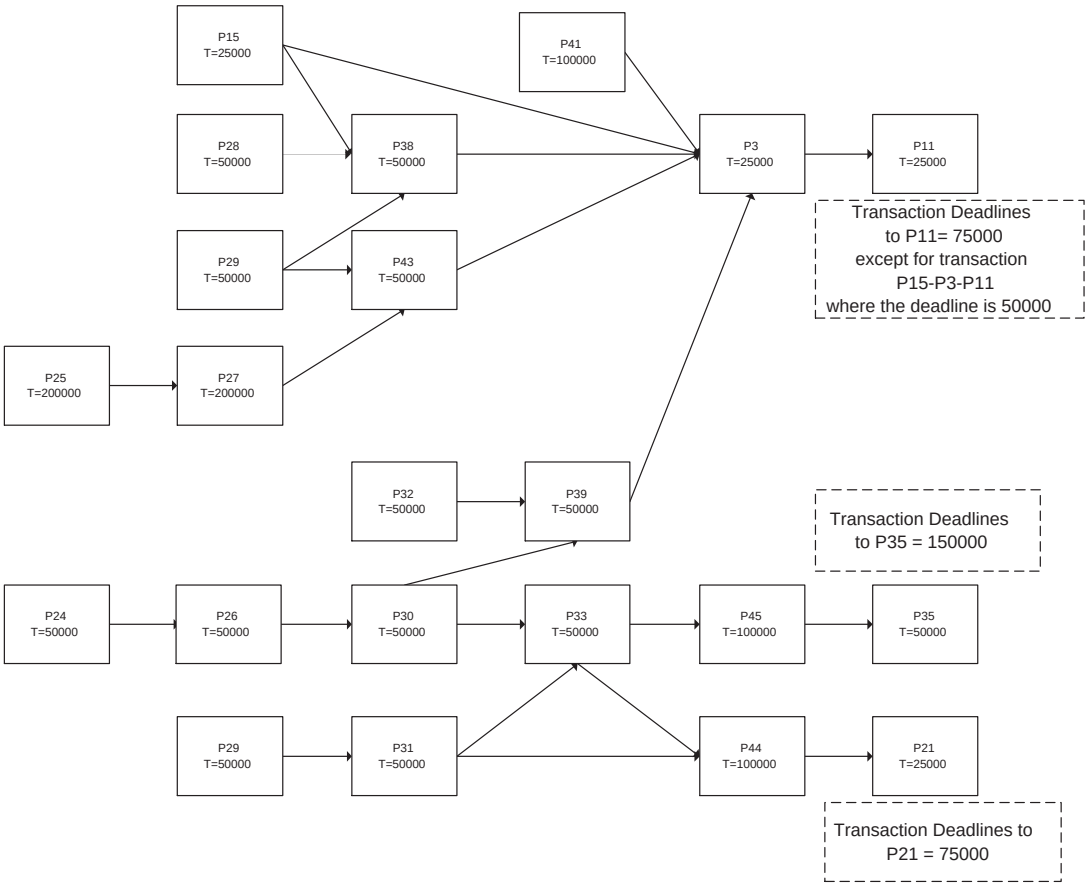


Figure 7.2: Diagram to Illustrate the System's Transactions Requirements

in the area of maintenance. However a greater benefit may be the more flexible scheduling model allowing valuable resources to be saved.

Consider the problem of implementing a cyclic scheduler for the system in this case study. The difficulty in synthesising the scheduler means that most of the tasks are split into small units of computation and some tasks into very small units. The fact that the major cycle rate is 200,000 means that another scheme is required to schedule the seven tasks ($P65-P71$) with an update rate of 1,000,000. One task executing every 200,000 units is used to sub-schedule the seven tasks. Each of the tasks with an iteration rate of 1,000,000 is dispatched every fifth time the 200,000 sub-scheduling task is executed. The added complexity of having a multi-level scheduler helps illustrate the need for a more flexible scheduling policy.

Rolls-Royce's experience of the application suggests the task's periods could also be changed. Some tasks need to be executed more frequently, others less frequently. However, the requirements have been produced assuming a cyclic scheduler is used and it is deemed appropriate not to alter their nature for the purpose of this case study.

## 7.3 Technical Transition to Fixed Priority Scheduling

The purpose of this section is to show how the timing requirements in Figure 7.3 and Table 7.2 can be implemented using the fixed priority scheduling theory developed in the course of the thesis. This section is to examine the technical issues while section 7.4 investigates the lifecycle issues. There are four parts to the technical discussion; task attribute assignment, task release, timing watchdog and schedulability analysis.

### 7.3.1 Task Attribute Assignment

Table 7.2 presents the results of task attribute assignment. The attributes are calculated using Algorithm 4 described in Chapter 6. There are three basic categories of columns: timing requirements, calculated task attributes and schedulability analysis results. In addition, two tasks are synthesised to represent the

overheads, which are task $TW$ for the timing watchdog and task $clk$ for the clock overhead. However, these are the product of the timing analysis for the fixed priority scheduler and do not represent actual tasks. The columns of the table, additional to those already defined for Table 7.1 represent:

**Calculated Task Attributes**

$D'$ is the calculated value for the deadline
$O$ is the calculated value for the offset
$P$ is the priority, which is derived from the deadline $D'$ using DMPO

**Schedulability Analysis Results** - values are considered later in section 7.3.5

$R$ is the result of the schedulability analysis
$R'$ is the worst-case response time allowing for any offset, i.e. $R' = R + O$
$Met?$ indicates whether the timing requirement is met, i.e. $R' \leq D'$
$C'$ represents the actual worst-case execution time of tasks including any overheads

Tasks $P11$, $P35$ and $P21$ have non-zero offsets to meet their jitter requirements. For example, task $P11$ has a jitter requirement of 12500 and an initial deadline of 25000. Due to the fact there is no information about the tasks' best case execution times, then a value of zero is assumed for their best case response. Using Algorithm 2, task P11 has an available execution window $= D_{11} - J_{11} = 25000 - 12500 = 12500$. Therefore task $P3$ has its deadline modified to $D'_3 = 12500$, and task $P11$ has its offset modified to $O_{11} = 12500$ and its deadline to $D'_{11} = 25000$. The deadlines are derived using Algorithm 2 that is given in Chapter 6. Clearly, if the tasks' deadlines are met, then the jitter requirements are also met.

Many tasks have their attributes modified so that transactions requirements are met. Figure 7.3 is a modified version of Figure 7.2 that illustrates the system's transactions, but also includes the modified task attributes (D for deadline, O for offset, and J for jitter in **bold**). Consider the transaction involving tasks $P25$, $P27$, $P43$, $P3$ and $P11$ (in that order), which has a transaction deadline of 75000. Figure 7.4 presents a time-line to illustrate the worst-case execution before the deadlines are modified. Clearly, the response time of the transaction is

| ID | T | C | C' | D' | D | O | P | R | R' | Met? |
|---|---|---|---|---|---|---|---|---|---|---|
| TW | 25000 | N/A | 100 | 25000 | 25000 | 0 | 1 | 100 | 100 | Y |
| clk | 25000 | N/A | 3500 | 25000 | 25000 | 0 | 2 | 3600 | 3600 | Y |
| P11 | 25000 | 671 | 771 | 25000 | 25000 | 12500 | 3 | 9411 | 21911 | Y |
| P21 | 25000 | 684 | 784 | 25000 | 25000 | 12500 | 4 | 10195 | 22695 | Y |
| P35 | 50000 | 173 | 273 | 50000 | 50000 | 37500 | 5 | 10468 | 47968 | Y |
| P3 | 25000 | 461 | 461 | 25000 | 12500 | 0 | 6 | 10929 | 10929 | Y |
| P1 | 25000 | 300 | 300 | 25000 | 25000 | 0 | 7 | 11229 | 11229 | Y |
| P2 | 25000 | 2088 | 2088 | 25000 | 25000 | 0 | 8 | 13317 | 13317 | Y |
| P4 | 25000 | 340 | 340 | 25000 | 25000 | 0 | 9 | 13657 | 13657 | Y |
| P5 | 25000 | 7 | 7 | 25000 | 25000 | 0 | 10 | 13664 | 13664 | Y |
| P6 | 25000 | 85 | 85 | 25000 | 25000 | 0 | 11 | 13749 | 13749 | Y |
| P7 | 25000 | 1910 | 1910 | 25000 | 25000 | 0 | 12 | 15659 | 15659 | Y |
| P8 | 25000 | 1971 | 1971 | 25000 | 25000 | 0 | 13 | 17630 | 17630 | Y |
| P9 | 25000 | 640 | 640 | 25000 | 25000 | 0 | 14 | 18270 | 18270 | Y |
| P10 | 25000 | 17 | 17 | 25000 | 25000 | 0 | 15 | 18287 | 18287 | Y |
| P12 | 25000 | 103 | 103 | 25000 | 25000 | 0 | 16 | 18390 | 18390 | Y |
| P13 | 25000 | 203 | 203 | 25000 | 25000 | 0 | 17 | 18593 | 18593 | Y |
| P14 | 25000 | 26 | 26 | 25000 | 25000 | 0 | 18 | 18619 | 18619 | Y |
| P15 | 25000 | 14 | 14 | 25000 | 25000 | 0 | 19 | 18633 | 18633 | Y |
| P16 | 25000 | 408 | 408 | 25000 | 25000 | 0 | 20 | 19041 | 19041 | Y |
| P17 | 25000 | 278 | 278 | 25000 | 25000 | 0 | 21 | 19319 | 19319 | Y |
| P18 | 25000 | 190 | 190 | 25000 | 25000 | 0 | 22 | 19509 | 19509 | Y |
| P19 | 25000 | 32 | 32 | 25000 | 25000 | 0 | 23 | 19541 | 19541 | Y |
| P20 | 25000 | 228 | 228 | 25000 | 25000 | 0 | 24 | 19769 | 19769 | Y |
| P22 | 25000 | 273 | 273 | 25000 | 25000 | 0 | 25 | 20042 | 20042 | Y |
| P23 | 25000 | 1265 | 1265 | 25000 | 25000 | 0 | 26 | 21307 | 21307 | Y |
| P24 | 50000 | 318 | 318 | 49997 | 50000 | 0 | 27 | 21625 | 21625 | Y |
| P25 | 100000 | 1334 | 1334 | 49998 | 100000 | 0 | 28 | 22959 | 22959 | Y |
| P26 | 50000 | 52 | 52 | 49998 | 50000 | 0 | 29 | 23011 | 23011 | Y |
| P27 | 200000 | 796 | 796 | 49999 | 200000 | 0 | 30 | 23807 | 23807 | Y |
| P28 | 50000 | 336 | 336 | 49999 | 50000 | 0 | 31 | 24143 | 24143 | Y |
| P29 | 50000 | 408 | 408 | 49999 | 50000 | 0 | 32 | 24551 | 24551 | Y |
| P30 | 50000 | 798 | 798 | 49999 | 50000 | 0 | 33 | 41343 | 41343 | Y |
| P31 | 100000 | 457 | 457 | 49999 | 100000 | 0 | 34 | 41800 | 41800 | Y |
| P32 | 50000 | 351 | 351 | 49999 | 50000 | 0 | 35 | 42151 | 42151 | Y |
| P33 | 50000 | 390 | 390 | 50000 | 50000 | 0 | 36 | 42541 | 42541 | Y |
| P34 | 50000 | 201 | 201 | 50000 | 50000 | 0 | 37 | 42742 | 42742 | Y |
| P36 | 50000 | 925 | 925 | 50000 | 50000 | 0 | 38 | 43667 | 43667 | Y |
| P37 | 50000 | 321 | 321 | 50000 | 50000 | 0 | 39 | 43988 | 43988 | Y |
| P38 | 50000 | 1801 | 1801 | 50000 | 50000 | 0 | 40 | 45789 | 45789 | Y |
| P39 | 50000 | 522 | 522 | 50000 | 50000 | 0 | 41 | 46311 | 46311 | Y |

| ID | T | C | C' | D | D' | O | P | R | R' | Met? |
|---|---|---|---|---|---|---|---|---|---|---|
| P40 | 50000 | 256 | 256 | 50000 | 50000 | 0 | 42 | 46567 | 46567 | Y |
| P41 | 100000 | 196 | 196 | 50000 | 100000 | 0 | 43 | 46763 | 46763 | Y |
| P42 | 50000 | 900 | 900 | 50000 | 50000 | 0 | 44 | 47663 | 47663 | Y |
| P43 | 50000 | 1945 | 1945 | 50000 | 50000 | 0 | 45 | 49608 | 49608 | Y |
| P44 | 100000 | 528 | 528 | 62500 | 100000 | 0 | 46 | 91921 | 91921 | Y |
| P45 | 100000 | 551 | 551 | 100000 | 100000 | 0 | 47 | 92472 | 92472 | Y |
| P46 | 100000 | 272 | 272 | 100000 | 100000 | 0 | 48 | 92744 | 92744 | Y |
| P47 | 100000 | 271 | 271 | 100000 | 100000 | 0 | 49 | 93015 | 93015 | Y |
| P48 | 100000 | 378 | 378 | 100000 | 100000 | 0 | 50 | 93393 | 93393 | Y |
| P49 | 100000 | 107 | 107 | 100000 | 100000 | 0 | 51 | 93500 | 93500 | Y |
| P50 | 100000 | 217 | 217 | 100000 | 100000 | 0 | 52 | 93717 | 93717 | Y |
| P51 | 100000 | 4698 | 4698 | 100000 | 100000 | 0 | 53 | 98415 | 98415 | Y |
| P52 | 100000 | 232 | 232 | 100000 | 100000 | 0 | 54 | 98647 | 98647 | Y |
| P53 | 100000 | 30 | 30 | 100000 | 100000 | 0 | 55 | 98677 | 98677 | Y |
| P54 | 100000 | 763 | 763 | 100000 | 100000 | 0 | 56 | 99440 | 99440 | Y |
| P55 | 100000 | 62 | 62 | 200000 | 200000 | 0 | 57 | 99502 | 99502 | Y |
| P56 | 200000 | 304 | 304 | 200000 | 200000 | 0 | 58 | 99806 | 99806 | Y |
| P57 | 200000 | 336 | 336 | 200000 | 200000 | 0 | 59 | 193408 | 193408 | Y |
| P58 | 200000 | 100 | 100 | 200000 | 200000 | 0 | 60 | 193508 | 193508 | Y |
| P59 | 200000 | 8 | 8 | 200000 | 200000 | 0 | 61 | 193516 | 193516 | Y |
| P60 | 200000 | 378 | 378 | 200000 | 200000 | 0 | 62 | 193894 | 193894 | Y |
| P61 | 200000 | 38 | 38 | 200000 | 200000 | 0 | 63 | 193932 | 193932 | Y |
| P62 | 200000 | 428 | 428 | 200000 | 200000 | 0 | 64 | 194360 | 194360 | Y |
| P63 | 200000 | 2258 | 2258 | 200000 | 200000 | 0 | 65 | 196618 | 196618 | Y |
| P64 | 200000 | 328 | 328 | 200000 | 200000 | 0 | 66 | 196946 | 196946 | Y |
| P65 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 67 | 196946 | 196946 | Y |
| P66 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 68 | 394692 | 394692 | Y |
| P67 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 69 | 399732 | 399732 | Y |
| P68 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 70 | 597078 | 597078 | Y |
| P69 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 71 | 794424 | 794424 | Y |
| P70 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 72 | 799464 | 799464 | Y |
| P71 | 1000000 | 5040 | 5040 | 1000000 | 1000000 | 0 | 73 | 996810 | 996810 | Y |

Table 7.2: Task Attributes and Schedulability Analysis Results

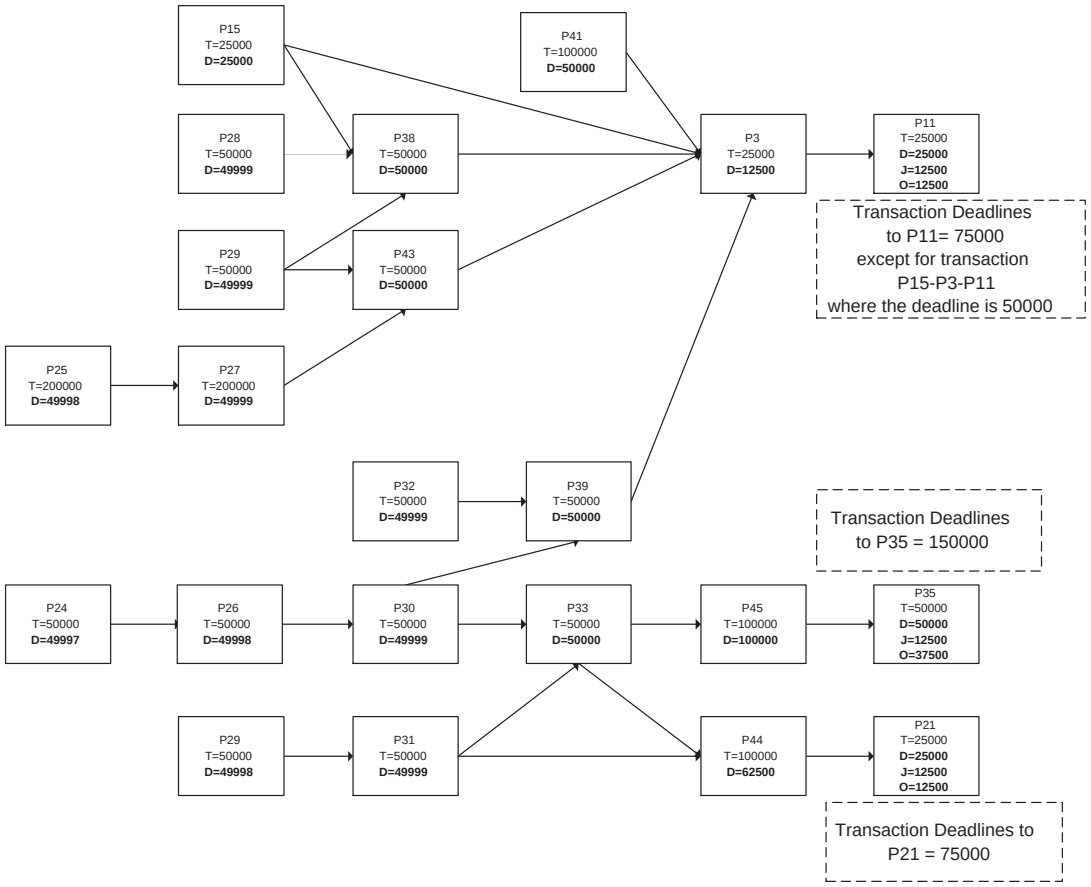


Figure 7.3: Diagram to Illustrate the Attributes of the Tasks in the Transactions

significantly greater than required. The tasks $P25$ and $P27$ have their deadlines reduced significantly (to 49998 and 49999 respectively). The net result of the deadline reduction is tasks $P25$ and P27 always precede task $P43$ (which has a deadline of 50000) so the end-to-end response of tasks $P25$, $P27$ and $P43$ is 50000. The remaining tasks ($P3$ and $P11$) have an end-to-end response of 25000 and periods of 25000.

Figure 7.5 presents a time-line to illustrate the worst-case execution after the deadlines are modified. Comparison of Figures 7.4 and 7.5 shows how the modified task attributes leads to the requirement being met with a significant reduction in response time. Figure 7.5 also shows how easy it is to demonstrate the requirements are met. It should be noted that in Figures 7.4 and 7.5 the instance of interest is represented by a shaded box.

The offsets and deadlines for tasks P3 and P11 are chosen so that task $P11$ meets its deadline and $P3$ precedes it. It should be noted that if $P3$ had a

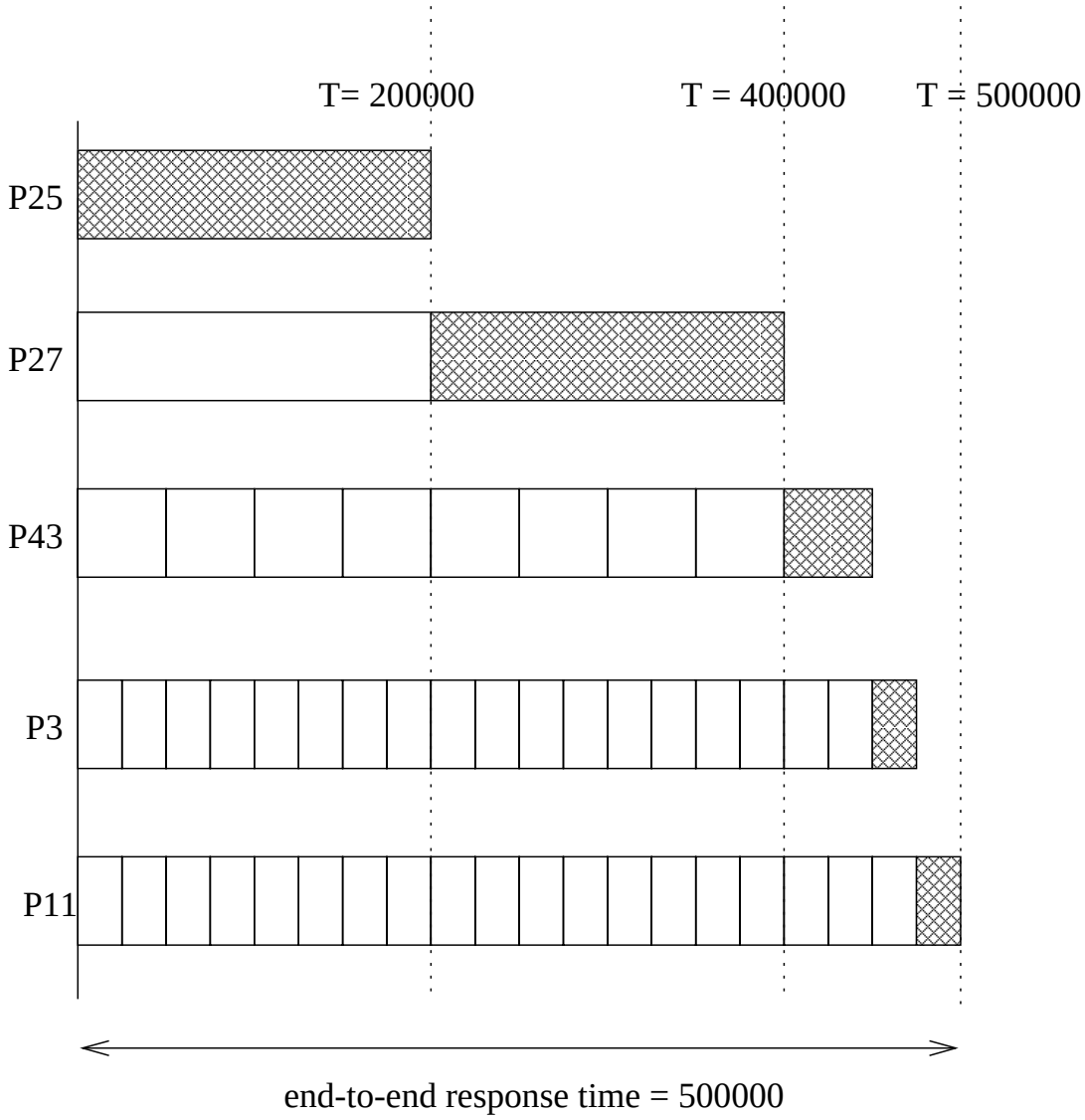


Figure 7.4: A Time Line for the Transaction Involving Tasks $P25$, $P27$, $P43$, $P3$ and $P11$ before the Attributes are Modified

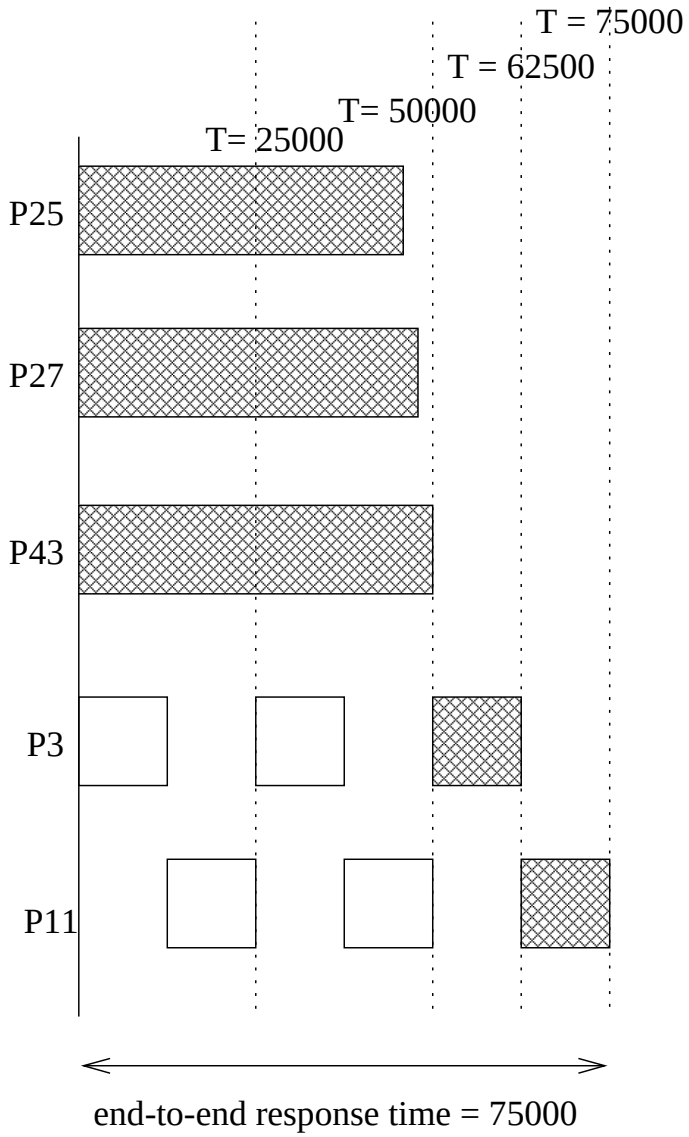


Figure 7.5: A Time Line for the Transaction Involving Tasks $P25$, $P27$, $P43$, $P3$ and $P11$ after the Attributes are Modified

larger deadline, then the ordering between $P3$ and $P11$ could not be guaranteed. Therefore, the transaction response time is 75000, i.e. the deadline is met.

This section has shown how complex system timing requirements can be broken down into task attributes using the technique developed in Chapter 6.

### 7.3.2 Task Release

The purpose of this section is to show how the specific tasks in Figure 7.3 and Table 7.2 should be released using the details in section 5.1 on how the hybrid task release mechanism works and how it can be analysed as a basis. The hybrid approach releases the majority of tasks based on a clock tick, with a few carefully selected tasks released in a time driven manner. The tasks released by the time driven approach are those whose rate is not a harmonic of the clock tick rate. The system already has a 25000 clock tick left over from the cyclic scheduler. Therefore, from a reuse perspective as many tasks as possible are released using the available clock tick. The overheads associated with the approach are:

$$\begin{aligned} C_{first} &= 100 \\ C_{sub} &= 50 \\ C_{TW} &= 100 \end{aligned}$$

where $C_{first}$ is the cost of releasing the first task,
$C_{sub}$ is the cost of releasing the subsequent tasks, and
$C_{TW}$ is the worst-case execution time of the timing watchdog software.

From Table 7.2, it can be seen that based on the tasks' periods, all the tasks can be released with zero jitter. However, Table 7.2 shows that tasks $P11$, $P21$ and $P35$ have non-zero offsets, which means the tasks would suffer release jitter of 12500, 12500 and 37500 respectively. Using the hybrid scheduling approach, tasks $P11$, $P21$ and $P35$ would be released in a time driven manner, whilst all the other tasks would be released in a tick driven manner. The tasks $P11$, $P21$ and $P35$ would have their execution time increased by $C_{first}$ to account for the overheads of the time driven release. Column C' of Table 7.2 represents the actual worst-case execution time of tasks that includes the overheads.

Table 7.2 also features a task, $clk$. The task, $clk$, is used to represent the clock overheads of all tasks except $P11$, $P21$ and $P35$. Task $clk$ has the following characteristics: $T_{clk} = 25000$ and $C_{clk}$ is calculated using equation (3.11) as shown in equation (7.1).

$$C_{clk} = 100 + (69 - \left\lceil \frac{25000}{25000} \right\rceil)50 = 3500 \tag{7.1}$$

This section has shown how system timing requirements can be supported with the infrastructure available using the technique developed in Chapter 5. In practice, the flexibility of the scheduling model could lead to the system engineers wanting to release tasks at other rates. An engineering guess could be taken of the likely changes, in fact Rolls-Royce has used the work in this thesis and changed some of the requirements. However, if the new requirements were used, then the ability to make direct comparisons between the cyclic scheduler and fixed priority scheduler would be lost. Early experimentation with iteration rates has indicated that adjusting tasks' periods can provide two benefits:

1. When the period is increased, then resources are saved. The precise saving depends on the actual application. However as an example, for the application discussed in this chapter about 20% of the processor utilisation could be saved.

2. In some cases where stability with the cyclic scheduler is marginal, the tasks' periods were increased, and jitter requirements imposed, with significant improvement in the stability obtained.

### 7.3.3 Timing Watchdog

The purpose of this section is to present how timing overruns are detected. To enable reuse, it is assumed the tick driven watchdog described in section 5.2.1 is used. The technique used to detect whether overruns have occurred is to check that the expected tasks have executed between clock ticks. The expected tasks are those with periods and deadlines less than or equal to the clock tick period. The overheads due to the timing watchdog in Table 7.2 are represented by task $TW$, which has characteristics of $C_{TW} = 100$ and $T_{TW} = 25000$. Tasks $clk$ and $TW$ have the highest priorities in the overall task set.

### 7.3.4 Implementation Details

This section is to discuss how the scheduler was implemented. A key fact about the implementation is that a hybrid scheduler is to be used. The fixed priority scheduler has two queues used for the controlled release and dispatch of tasks. The run queue contains the tasks that are runnable. The delay queue contains the tasks awaiting release. At the end of each task's execution, the run queue is updated and the highest priority task is executed.

Figure 7.6 illustrates how the task dispatch mechanism is implemented. A key feature of the implementation is the search mechanism that is employed to find the highest priority task. The search for the highest priority task is started from the last task executed unless tasks have just been released, in which case the search starts at the overall highest priority task. The reason for this approach is that the search time should be reduced.

The method employed to release tasks from the delay queue to the run queue is to define a number of words each of which is N bits long (where N is the number of tasks). Each task is allocated a position in the word based on its priority, i.e. the highest priority task relates to the least significant bit. A value of logic '1' would mean a task is active (i.e. runnable for the run queue, or releasable for the delay queue), and a logic '0' otherwise. Figure 7.7 illustrates the format of the words.

The words defined are the run queue and delay queue words. Each delay queue word represents the set of tasks that have identical release points rather than a word for each task in order to reduce the number of operations the scheduler must perform. Associated with each delay queue word (in Figure 7.6 the word is referred to as "TimeToRun") is an iteration rate and a word that indicates the number of clock ticks before its task set is runnable again. When the task is due to be released again, the run queue is OR'ed with the relevant delay queue word to produce a new run queue word.

Figure 7.6 also shows how the run queue is updated upon a regular clock tick, which is also used to trigger the timing watchdog. Many of the implementation decisions taken were influenced by the need to implement the system in SPARK Ada [92]. For instance, fixed sized arrays are used to represent the

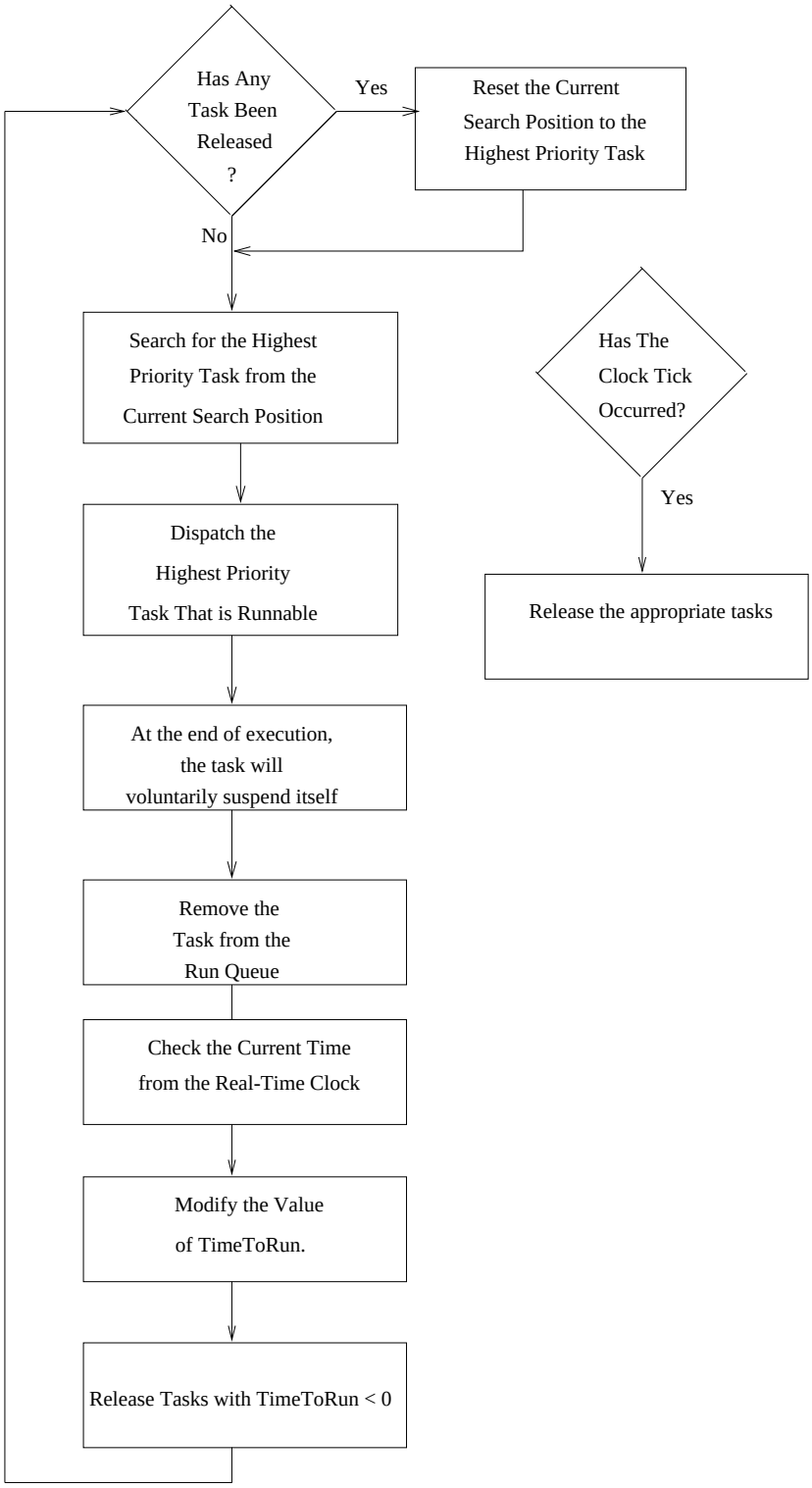


Figure 7.6: Operation of the Hybrid Scheduler

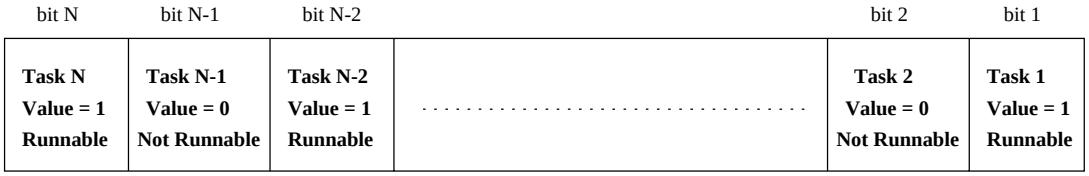


Figure 7.7: Diagram to Illustrate the Word Format

queues whereas, in systems that are not deemed to be critical dynamic data allocation to memory is often used.

### 7.3.5 Schedulability Analysis Results

Table 7.2 also presents the results from the schedulability analysis. The worst case response time is presented in two columns: $R$ is the result of the schedulability analysis assuming a critical instant, and $R'$ is a modified value that accounts for a task's offset, i.e. $R' = R + O$. The tasks' response times in Table 7.2 are less than their deadlines, which indicates that the task set is schedulable. This is despite an overall processor utilisation of 99.68%, which is almost the maximum possible. Hence, it can be concluded the system's timing requirements are met.

The time taken to derive the task attributes by hand was less than five minutes. The schedulability analysis could have been performed in less than an hour, however an automatic tool was employed that repeated the task attribute assignment and performed the schedulability analysis. The tool completed the job virtually instantaneously. The ease in which the attribute assignment and subsequent analysis may be performed manually is an indication that the *understanding* criterion is met.

With respects to task attribute assignment, the case study has demonstrated the technique in Chapter 6 is capable of calculating attributes for a real complex system. This does not prove the sufficiency criterion is met, but suggests that it is likely to be satisfied. Rolls-Royce's engineers have used the technique successfully by hand, and have also produced a scheduler and tool [96] that is intended to be compliant with DO-178B [33]. Hence, it can also be stated the certification criterion is met as well as the approach being considered understandable. The key fact is that all the existing software and hardware, except the scheduler, is reused - this is a strong indication that the approach meets one of its principle

objectives. However, the reuse criterion is not entirely met since any tools for producing cyclic schedulers cannot be reused.

## 7.4 Details of the Effect on the Process

Section 2.3 describes a typical process currently used in industry for designing safety critical systems based on cyclic schedulers. The purpose of this section is to consider how a move to fixed priority scheduling would affect the way in which the system is produced. The consideration is based on the following stages of the life-cycle:

1. Requirements
2. Design
3. Implementation
4. Verification
5. Certification

### 7.4.1 Requirements

The general nature of the requirements captured is not altered because of a change in scheduling regime. The reason is the requirements are dependent on the application, the design and the domain. The scheduler is simply a means to provide timeliness. The only requirements that do change is the functional requirements for the scheduler itself that are expected to change. Experience shows the change to a more flexible scheduling regime may cause an impact on the requirements being written. The requirements need to be more stringent whilst taking advantage of the improved flexibility of the scheduling model.

The requirements need to be more stringent because the static nature of the current scheduler leads to many requirements being implicit rather than explicit in the implementation. For example, a task may only work if it has low jitter and so in a cyclic schedule it is always placed at the start of a minor cycle. To implement the task successfully with fixed priority scheduling necessitates

the jitter constraint to be explicitly stated. Similarly, precedence requirements or separation requirements may be implicit in the implementation of a cyclic scheduler because of the static control flow. With a fixed priority scheduler the requirements need to be explicitly stated.

The increased flexibility of the scheduler causes problems because the engineers have to determine what the real requirements are. Also, the existing requirements are derived and specified based on the knowledge that only harmonic iteration rates of the clock tick are achievable. A good example is the tasks' periods that are chosen. Engineers know that the rate for a particular task is not 25000, but they know that 25000 works. Therefore, the existing requirements tend to be largely historical, the reasoning for which may be poorly understood. This means the system almost has to be re-engineered to establish what the real requirements are. "New" requirements can be established through analysis, simulation and testing.

The problems caused by having to establish the "real" set of requirements are:

1. Changing the requirements affects reuse because the system's basis is different. Therefore, a certain amount of re-verification is necessary.

2. Part of the certification argument is based on the confidence that the timing requirements have been used on generations of engine controller. Therefore, an argument has to be produced that the change of requirements has a positive effect rather than negative effect, i.e. the resulting system is at least as safe.

3. A control system is complex and difficult to analyse, therefore establishing the "real" requirements may be difficult. A great deal of effort and cost cannot be committed unless real benefits are obtainable. It should be noted that this is likely to be a one-off cost.

4. Experience has shown that in practice engineers will decide the functionality should execute faster, particularly for outputs such as actuators [96]. Therefore, more processing resource rather than less is liable to be demanded. Hence, a benefit of fixed priority scheduling could easily be lost (i.e. helping to reduce processor utilisation by providing greater flexibility

in the choice of tasks' periods) even though a better control system may result.

The single biggest impact of a change to a more flexible scheduling is the need to re-assess what the requirements are. However, there are also considerable benefits in doing this since the system should perform better and safer if the requirements are correct. In addition, it should be possible to reduce the amount of testing that is necessary. The reason is a great deal of testing is performed to check the system's operation even though the requirements may already be proven correct.

### 7.4.2 Design

The introduction of fixed priority scheduling has two implications for the design process; task attributes have to be assigned that meet the system's timing requirements and a new scheduler is needed. Chapter 6 already describes how task attributes are assigned, and section 7.3.1 further demonstrates the point. To design a kernel requires the consideration of the top level requirements, these are:

1. the scheduler is non-preemptive,
2. some tasks are released in a tick driven manner and others in a time driven manner, and
3. the software for the tick driven timing watchdog is to correctly determine whether the system's operation is proceeding as planned.

The scheduler should be designed in a manner consistent with safe programming practices, such as those advocated by SPARK Ada [92]. For instance all memory usage must be decided before hand in a static fashion, which excludes the use of pointers. Pointers are often used in the design of the scheduler, since the task information must be stored dynamically because the number of tasks is unknown. However in a safety critical system, the number of tasks must be known before hand. Therefore, it is much easier to use static data structures in this case rather than in a system that has a more dynamic nature.

### 7.4.3 Implementation

The change to a fixed priority scheduler causes two changes to the implementation stage of the life-cycle. These relate to the scheduler software and the worst-case execution time of tasks. The scheduler implementation is to meet its design taking into account the need to use a safe subset of a language. After the implementation is produced, worst-case execution time analysis should be performed as described in section 2.3.4.

The worst-case execution time analysis is performed for two reasons, to allow schedulability analysis to take place and to establish whether the execution budgets are not exceeded. Execution budgets are often assigned during the requirements phase to help manage resource usage. The impact on worst-case execution time analysis is that with fixed priority scheduling there is a greater emphasis placed on analysis. In contrast, the verification of a cyclic scheduler generally relies on test. The use of analysis also means greater importance is attached to having correct worst-case execution times.

### 7.4.4 Verification

The single biggest change when adopting a fixed priority scheduler is that there is no longer a static ordering of tasks. Section 7.4.1 has already discussed how a dynamic task ordering means more stringent requirements are necessary. The problem caused for verification is related to the fact the run-time ordering is no longer deterministic. Therefore, the timing requirements cannot be verified using a test-based approach. To accommodate the change in scheduling policy, the verification strategy is changed so that the primary evidence of timing correctness is obtained through the timing analysis. Next, a reduced amount (relative to how much was performed with the cyclic scheduler) of test is used to provide extra confidence of the correctness of the system's timing properties. Hence, it can be stated the scheduling is at least as predictable with fixed priority scheduling.

When the change to fixed priority scheduling was raised with the Joint Airworthiness Authority (JAA), it was met with approval because analysis was viewed as being better than testing [96]. Determinism is only stipulated because the current process is based on a test based philosophy. Therefore, the ability to

observe how the system operates is important. An argument can be used that the worst-case is deterministic (i.e. the schedule of task can be "laid out") even though the typical cases are not. The production of time-lines demonstrates this point. It can be stated that the worst-case fixed priority schedule is analogous to that of the cyclic scheduler.

The functional verification of the system should not be affected if all the necessary timing requirements, such as precedence relations, are specified. However, if all the necessary requirements are not specified, then anomalies may arise when the task ordering changes. In terms of testing, there is a slight increase in the number of paths through the software, however these are restricted to the module that contains the scheduler. Therefore, there is a slight increase in the amount of testing necessary. Part of the functional verification of the system is to show that the scheduler is correctly implemented. One of the tests includes showing that periodic tasks are released at the correct rate with the specified offset.

### 7.4.5 Certification

There are two distinct parts to the certification stage of the life-cycle. These parts are; structuring the safety argument to justify the system is safe and showing that no additional hazards are introduced by the change. Structuring the safety argument is considered to be out of the scope of this work. However, there is a great deal of work on how the argument may be structured, an example of which is found in [104].

The interesting challenge is to argue that no additional hazards are introduced, i.e. the system is at least as safe as before the change. It can be stated that in fault-free conditions the ability to guarantee the timing requirements are met is increased through using analysis instead of test. Therefore, the problem becomes a matter of showing the fault tolerance (i.e. ability to deal with timing overruns) is at least as effective as with the cyclic scheduler. Also, the impact of any change to the interface with the rest of the system has to be assessed.

To show that the fault tolerance, with respect to scheduling, is unaffected requires a demonstration that timing overruns are detected at least as efficiently as with the cyclic scheduler. It is assumed that the fault recovery strategy initially remains unchanged. Section 5.2 contains a discussion of the various techniques

(including their pros and cons) for implementing a timing watchdog for a fixed priority scheduler. The principal technique for fault recovery is to have a lane change followed by a reset of the faults lane. Two main techniques are considered; the tick driven watchdog and the countdown timer watchdog. The conclusion is that either technique is at least as good at detecting timing overruns as the timing watchdog of the cyclic scheduler.

There are two aspects to the interaction with the rest of the system; the time ordering of tasks and their interface. The fact that tasks are executed in a different order has already been dealt with. The interface between the rest of the system and the scheduler is unaffected because:

1. there is no need to alter the way in which call-outs to tasks are handled, i.e. exactly the same variables can be passed and the actual procedure calls can be made in the same way, and

2. the scheduler interface to the timing watchdog, including event handling, is the same.

The fact that the interface to the rest of the system is unaffected and the proof of scheduling correctness is now based on mathematical evidence, this means the certification evidence derived is in fact improved. As already stated in section 7.4.4, when consulted the relevant certification authority confirmed this opinion.

## 7.5 Summary

The contributions of this chapter are to demonstrate:

1. How the scheduling techniques described in Chapters 3 - 6 relate to a change of scheduling policy in a real system. The work has shown that the techniques are sufficient to handle the complexity and provide definitive results whilst being comprehensible.

2. The impact on the process is the need for the actual "real" requirements to be expressed that can then be guaranteed as being met through analysis. Combined with improved certification evidence through using analysis,

rather than test, the change adds a great deal of benefits to the final product.

The chapter has helped to show the implications to the product and process lifecycle of a change from a cyclic scheduler to a fixed priority scheduler for a real system. The fixed priority scheduling technique has been used by Rolls-Royce on the actual engine controller. During trials, no problems were found with the engine controller after the change [96].

# Chapter 8

# Analysis of Task Sets That Feature Offsets

The purpose of this chapter is to address the issue of timing analysis for task sets that feature offsets. The majority of the timing analysis published to date and the contents of this thesis has assumed a critical instant. Liu and Layland [10] state that for task sets that feature no offsets there is a critical instant where all tasks are simultaneously released. In these circumstances, there are a variety of tests exist that are both sufficient and necessary.

However, in practical systems it is sometimes necessary to offset the execution of tasks from one another. For these cases, the test based on a critical instant is pessimistic because the analysis does not account for the phasing of tasks [51]. Offsets are used within real systems so that actual requirements and design-derived requirements can be met. These include:

1. *Actual Requirement*: Ensuring a particular action has been performed before releasing another task. For example, task A may request data from a hardware device, however, the data may not be available until a later time. Therefore, an associated task B is required to read the data. Hence the task B would have to be offset by an appropriate amount. The approach in Algorithm 3 of Chapter 6 uses offsets to enforce separation requirements.

2. *Design-Derived Requirement (A)*: The spreading of processor resource requirements through a given time frame so that jitter can be reduced for a

specific task. To achieve a jitter requirement, a task's deadline is reduced. The effect of this change is the task may have a higher priority, which could increase the interference on lower priority tasks. To prevent too great an impact on other tasks, the task is phased to occur at an appropriate time. The task attribute assignment approach in Chapter 6 uses offsets as part of Algorithm 2 for dealing with jitter requirements. The timing requirements for tasks $P11$, $P21$ and $P35$ in the case study of Chapter 7 presents an example of this nature.

3. *Design-Derived Requirement (B)*: Ensuring precedence relations are maintained by not allowing a task to be released until another has completed its execution. Section 3.2.2 has already shown how offsets may be used for enforcing precedence as part of a distributed scheduling mechanism. Chapter 9 makes similar use of offsets.

A useful observation is that all of these requirements are flexible in nature. For example consider a separation requirement between two tasks. If the minimum separation between the two tasks is always maintained and the second task meets its required deadline, then the actual offset does not matter. The approach taken within this chapter utilises the flexibility available.

Section 3.1.6 of the literature survey describes existing approaches that have been attempted to analyse tasks with non-zero offsets. However, a number of problems were found with these approaches. Therefore, this chapter is to investigate new techniques for solving the problem that are optimised (in the context of pessimism, understandable and computational complexity) for the types of application expected.

Section 8.1 discusses the existing approach where task offsets are ignored and it is assumed there is a critical instant. Section 8.2 discusses the exact analysis approach, where each individual task release is checked for schedulability for the minimum period that the execution order takes to repeat. Section 8.3 presents the new approach that entails forming a composite task with no offset to represent the tasks with offsets. Section 8.4 investigates how effective the composite approach is when compared to the critical instance and exact approaches. Finally, section 8.5 investigates how further improvements to the composite model may be provided.

## 8.1 Analysis of Task Sets That Feature Offsets Assuming a Critical Instant

Simple analysis of task sets with offsets can be carried out by effectively ignoring the offsets and assuming a single critical instant. Each task in the set of tasks is transformed as illustrated in equations (8.1) and (8.2). The transformation has no effect on tasks already having a zero offset.

$$D'_i = D_i - O_i \tag{8.1}$$

where $D'_i$ is the modified value of task $i$'s offset, and
$D_i$ is the original value of task $i$'s offset

$$O_i = 0 \tag{8.2}$$

The main problem with this approach is that the analysis is especially pessimistic since the ability to balance resources is lost. Consider the "Design-Derived Requirement (A)" (from the list at the start of the chapter) where tasks with tight jitter requirements have to be phased. The analysis would effectively ignore the phasing. The task set characteristics outlined in Table 8.1 are presented as an example.

| Id | T | O | D | C |
|---|---|---|---|---|
| A | 25000 | 0 | 6000 | 2000 |
| B | 25000 | 6250 | 12000 | 1500 |
| C | 25000 | 13000 | 18000 | 1500 |
| D | 25000 | 18000 | 25000 | 1500 |
| E | 50000 | 0 | 50000 | 2000 |
| F | 100000 | 0 | 100000 | 1000 |
| G | 200000 | 0 | 200000 | 1000 |
| H | 1000000 | 0 | 1000000 | 2500 |

Table 8.1: Example task set

Table 8.1 has five columns, where the columns represent:

$Id$ is the identifier of the task
$T$ is the period of the task
$O$ is the offset of the task
$D$ is the deadline of the task
$C$ is the worst-case execution time of the task

Table 8.2 contains the results of the schedulability analysis using the approach described in this section. The table illustrates that the resultant task set is not schedulable because Task $A$ misses its deadline. The columns of Table 8.2 are a superset of those for Table 8.1. The additional columns represent:

$R$ is the worst-case response time of the task
$Met$? indicates whether the task is schedulable, i.e. $R \leq D$

The analysis in Table 8.2 is based on the timing analysis for a non-preemptive scheduling model described in Chapter 4 and priorities assigned using the deadline monotonic priority ordering.

| Id | T | O | D | P | C | R | Met? |
|---|---|---|---|---|---|---|---|
| C | 25000 | 0 | 5000 | 1 | 1500 | 4000 | Y |
| B | 25000 | 0 | 5750 | 2 | 1500 | 5500 | Y |
| A | 25000 | 0 | 6000 | 3 | 2000 | 7500 | N |
| D | 25000 | 0 | 7000 | 4 | 1500 | 9000 | N |
| E | 50000 | 0 | 50000 | 5 | 2000 | 11000 | Y |
| F | 100000 | 0 | 100000 | 6 | 1000 | 12000 | Y |
| G | 200000 | 0 | 200000 | 7 | 1000 | 13000 | Y |
| H | 1000000 | 0 | 1000000 | 8 | 2500 | 13000 | Y |

Table 8.2: Results of the Simple Analysis

Table 8.2 helps to show how the simple form of analysis can lead to pessimistic results caused by the tasks with a period of 25000 not having their resource distributed over time. Tasks A and D fail to meet their deadline mainly due to the interference caused by the higher priority tasks B and C. However, inspecting

the task set shows the tasks A, B, C and D should not interfere with each other since their offsets and deadlines mean their executions cannot overlap. Therefore, more appropriate analysis should be found.

## 8.2 Exact Analysis

Section 3.1.2 of the literature survey first discusses exact analysis within this document. The exact analysis takes the approach of showing every individual release of a task is schedulable assuming all tasks execute for their worst-case execution time. The original consideration of timing analysis, presented by Leung and Whitehead [53] shows that if every instance of every task on a particular processor is schedulable over the period [Maximum Offset of Any Task, 2×Least Common Multiple of the Task Periods + Maximum Offset of Any Task) then the task set is schedulable. An instance of a task is defined as its release, execution and completion. Audsley [54] improved the analysis so the duration became [Maximum Offset of Any Task, Least Common Multiple of the Task Periods + Maximum Offset of Any Task).

The exact analysis approach does not suffer the pessimism of any of the other approaches, such as those discussed in section 3.1.2 and in section 8.1. The pessimism of the other approaches originates in two areas. Firstly, the analysis assumes the task set has a single critical instant, which effectively removes the phasing provided by having offsets. Secondly, the blocking model for the timing analysis is based on the critical instant, which introduces pessimism since it is assumed every lower priority task can block the task being analysed. This is clearly pessimistic because it relies on the assumption that the task being analysed is runnable when any of the lower priority tasks is being run. There are many cases where this is not true.

Table 8.3 presents the results of the schedulability analysis derived using exact analysis for the task set in Table 8.1. The results show the task set is schedulable when analysed using the exact approach. In contrast, Table 8.2 demonstrates the task set is unschedulable using the approach that ignores offsets.

The problem with the exact analysis approach is that for task sets that feature co-prime periods then the number of releases that require checking may become

prohibitively large. Table 8.4 gives the number of releases for the example task set in Table 8.1. However, if 2 tasks are added to the task set with iteration rates of 97000 and 37000, then the number of releases to be checked becomes $(97 \times 37 \times 196) + (97 \times 196) + (37 \times 196) = 7029708$. The number of task instances that need to be checked could become large. If it is considered that the task sets typically contains upwards of fifty tasks, then the computational complexity may result in a problem that is effectively infeasible.

| Id | T | O | D | P | C | R | Met? |
|---|---|---|---|---|---|---|---|
| C | 25000 | 13000 | 18000 | 1 | 1500 | 14500 | Y |
| B | 25000 | 6250 | 13000 | 2 | 1500 | 10000 | Y |
| A | 25000 | 0 | 6000 | 3 | 2000 | 2000 | Y |
| D | 25000 | 18000 | 25000 | 4 | 1500 | 19500 | Y |
| E | 50000 | 0 | 50000 | 5 | 2000 | 4000 | Y |
| F | 100000 | 0 | 100000 | 6 | 1000 | 5000 | Y |
| G | 200000 | 0 | 200000 | 7 | 1000 | 6000 | Y |
| H | 1000000 | 0 | 1000000 | 8 | 2500 | 8500 | Y |

Table 8.3: Results of the Exact Analysis

| Id | T | O | Releases $\left(= \frac{LCM(T_i)}{T_i} = \frac{1000000}{T_i}\right)$ |
|---|---|---|---|
| A | 25000 | 0 | 40 |
| B | 25000 | 6250 | 40 |
| C | 25000 | 13000 | 40 |
| D | 25000 | 18000 | 40 |
| E | 50000 | 0 | 20 |
| F | 100000 | 0 | 10 |
| G | 200000 | 0 | 5 |
| H | 1000000 | 0 | 1 |
| Overall | | | 196 |

Table 8.4: Number of Task Releases to be Verified

## 8.3 A Composite Approach

Experience in performing timing analysis of real industrial systems leads to the observation that there is likely to be a pattern in the types of offset used. The use of offsets is likely to be analogous to that of the cyclic scheduler, i.e. the processing frame is split into manageable chunks with tasks allocated to the different partitions using offsets. Table 8.1 can be used to illustrate the type of requirement that result. In Table 8.1, tasks A, B, C and D have offsets such that their computation is spaced relatively evenly over the period of 25000 units.

The purpose of this section is to investigate how a timing analysis approach may be derived. The timing analysis is to be tailored for the type of timing requirements expected and is intended to be easily understood.

### 8.3.1 Defining the Composite Task

This section presents an approach based on the formation of composite tasks **that are used for analysis purposes only**, i.e. the composite task does not feature as part of the implementation at run-time. The motivation behind this approach is that if a composite task with zero offset can represent the tasks with non-zero offsets, then the analysis presented in section 3.1 may be used. The benefit of the composite task approach is that the computational complexity is kept sufficiently low, while allowing the resource to be spread through time. However, one of the principal benefits of the technique is the fact the existing analysis can be used with a limited amount of pre- and post-processing. There are a number of benefits of being able to reuse the analysis, including the fact exisiting tools and training can still be used as well as only a limited amount of extra certification evidence is needed.

The following is a number of steps to be followed to derive the composite task. The steps are backed up by a threaded example (presented in italics) based on the task set in Table 8.1.

1. A composite task is created for each period where there are tasks with non-zero offsets. To reduce pessimism one task is included in the set of tasks used to form the composite task. The one task has zero offset and

the same period as the rest of the tasks in the set. This one task can be chosen from any that meet the criteria of zero offset and the same period as the other tasks in the set. If a task with the same period and zero offset does not exist, then just the tasks with non-zero offsets are represented by the composite task.

*For the task set in Table 8.1, there is a need for one composite task to represent tasks B, C and D. These tasks have non-zero offsets and the same period of 25000. The composite task also represents task A that has the same period as tasks B, C and D, and a zero offset.*

2. For each composite task, define a set ST that consists of the tasks with the same period and non-zero offsets, and a maximum of one task (if one exists) with the same period and zero offset.

   *For the task set in Table 8.1, the set ST consists of four tasks B, C and D (by virtue of having the same period and non-zero offset), and task A (by virtue of having the same period and zero offset).*

3. For each composite task, define a set ST1 that consists of the tasks with the same period and non-zero offsets. If the composite task is to represent a task with zero offset, then an additional task is added to set ST1 with the same period as the other tasks and an offset equal to its period. The additional task is intended to represent the second instance of the task with zero offset. In other words the set ST1 represents the instances of the tasks in set ST in the time range (0, period of the tasks in set ST].

   *For the task set in Table 8.1, the set ST1 consists of four tasks B, C and D as well as one other denoted as X. Task X has a period equal to the period of tasks B, C and D (i.e. 25000) and an offset equal to its period.*

4. For each composite task, a worst-case execution time is assigned equal to the maximum execution time of any task in the set ST.

   *For the task set in Table 8.1, the composite task's worst case execution time is:*

$$\begin{aligned} C_{COMP} &= maximum\{C_A, C_B, C_C, C_D\} \\ &= maximum\{2000, 1500, 1500, 1500\} \\ C_{COMP} &= 2000 \end{aligned} \tag{8.3}$$

5. For each composite task, define a set ST2 which is an ordered version of ST1. The ordering is in accordance with increasing value of offset for the tasks in set ST1.

   *For the task set in Table 8.1, the set ST2 is defined as:*

   (a) *task B (offset is 6250) followed by*

   (b) *task C (offset is 13000) followed by*

   (c) *task D (offset is 18000) followed by*

   (d) *task X (offset is 25000).*

   i.e. $ST2 = \{B,\ C,\ D,\ X\}$

6. For each composite task, define a set ST3 where the members are ordered the same as set ST2. However, the value of the members is altered to be equal to the offset of the member in set ST2 divided by the index (in the range 1..N) into the set ST2.

   *For the task set in Table 8.1, the set r is defined as:*

$$\begin{aligned} ST3 &= \left\{ \frac{O_B}{\text{index of member B in set ST2}}, \frac{O_C}{\text{index of member C in set ST2}}, \right. \\ &\quad \left. \frac{O_D}{\text{index of member D in set ST2}}, \frac{O_X}{\text{index of member X in set ST2}} \right\} \\ &= \left\{ \frac{6250}{1}, \frac{13000}{2}, \frac{18000}{3}, \frac{25000}{4} \right\} \\ ST3 &= \{6250, 6500, 6000, 6250\} \end{aligned} \tag{8.4}$$

7. For each composite task, the task is given a period equal to the minimum value of any member in the set ST3.

   *For the task set in Table 8.1, the period of the composite task is the minimum of 6250, 6500, 6000 and 6250. Therefore, the period of the composite task is 6000.*

8. For each composite task, the task is given a deadline equal to the minimum relative deadline for any task in the set ST. The minimum relative deadline is classed as the task's deadline minus the task's offset.

*For the task set in Table 8.1, the composite task's deadline is equal to the minimum of:*

(a) $D_A - O_A = 6000$

(b) $D_B - O_B = 6250$

(c) $D_C - O_C = 5000$

(d) $D_D - O_D = 7000$

$\therefore D_{COMP} = 5000$

9. Replace the tasks that form the composite task (i.e. the members of set ST) with the composite task in the task set to be analysed.

   *For the task set in Table 8.1, tasks A, B, C and D are replaced with the task COMP. The resulting task set consists of the tasks COMP, E, F, G and H.*

10. Perform the "standard" schedulability analysis for the task set that includes the composite task.

    *Table 8.5 illustrates the results of the schedulability analysis. The results show that the task set is schedulable using the composite approach. Table 8.5 shows how tasks A, B, C and D are replaced by a single composite task with characteristics of: period of 6000, deadline of 5000 and worst-case execution time of 2000.*

11. For each task that is replaced by the composite task (i.e. the tasks that are members of set ST), determine the task's worst-case response time using equation (8.5).

$$R = O + R_{COMP} \tag{8.5}$$

*For the task set in Table 8.1, the worst-case response times are:*

$$\begin{aligned} R_A &= O_A + R_{COMP} = 0 + 4500 = 4500 \\ R_B &= O_B + R_{COMP} = 6250 + 4500 = 10750 \\ R_C &= O_C + R_{COMP} = 13000 + 4500 = 17500 \\ R_D &= O_D + R_{COMP} = 18000 + 4500 = 22500 \end{aligned} \tag{8.6}$$

| Id | T | O | D | P | C | R | Met? |
|---|---|---|---|---|---|---|---|
| COMP | 6000 | 0 | 5000 | 1 | 2000 | 4500 | Y |
| E | 50000 | 0 | 50000 | 2 | 2000 | 8500 | Y |
| F | 100000 | 0 | 100000 | 3 | 1000 | 9500 | Y |
| G | 200000 | 0 | 200000 | 4 | 1000 | 10500 | Y |
| H | 1000000 | 0 | 1000000 | 5 | 2500 | 10500 | Y |

Table 8.5: Results of the Composite Analysis

### 8.3.2 Proof of Correctness

Obviously, with any new approach to timing analysis it is important to show that it is a sufficient test. A sufficient test is where, if the test indicates that the system meets its requirements, then it will in practice. However, if the results suggest the system does not meet the requirement, then in practice the system may still do so. In this respect the test can be considered as pessimistic. Tindell [105] shows a test is sufficient but not necessary when the interference experienced by all lower priority tasks does not decrease when compared with another sufficient test. Therefore, to prove that the composite analysis is sufficient when compared with the "standard" schedulability analysis, the interference caused by the composite task and the tasks that form it need to be compared.

**Theorem 4.** *The interference caused by the composite task experienced by all lower priority tasks does not decrease compared to the interference caused by the tasks that form the composite task.*

#### Proof

The sufficient test to be used as a comparison is the exact analysis in section 8.2. Based on the definition of the composite task in section 8.3.1, it can be stated the composite task has a higher priority than any of the tasks formed from it. Therefore, if it can be shown the condition in equation (8.7) is true, then the test is sufficient. Equation (8.7) represents a statement that the maximum interference of the composite task is always greater than, or equal to the maximum interference of the tasks from which it is created. The left hand side of equation (8.7) is the interference caused by the composite task at any time $t$. The time $t$ is with respect to a reference time when all periodic tasks with zero offset are

simultaneously released. The right hand side of equation (8.7) is the interference of the tasks that form the composite task at any time $t$.

$$\forall t : \left\lceil \frac{t}{T_{COMP}} \right\rceil C_{COMP} \geq \sum_{j \in ST} \left| \left\lceil \frac{t - O_j}{T_k} \right\rceil \right|_0 C_j \tag{8.7}$$

where $T_k$ is the period of the tasks that are members of set ST

The symbol $||_0$ signifies a limiting function that constrains the answer to the set of non-negative numbers (i.e. any negative input returns a value of zero, positive numbers are unchanged). Therefore, the combination of symbols $|\lceil\rceil|_0$ means that the answer is always a whole number and non-integer numbers are rounded up.

Based on the definition of the composite task in section 8.3.1, it can be stated that $C_{COMP} \geq C_j$. Therefore, equation (8.7) can be simplified to produce equation (8.8).

$$\forall t : \left\lceil \frac{t}{T_{COMP}} \right\rceil \geq \sum_{j \in ST} \left| \left\lceil \frac{t - O_j}{T_k} \right\rceil \right|_0 \tag{8.8}$$

Based on the definition of the composite task in section 8.3.1, it can be stated that the tasks from which the composite task is formed repeat their execution every $T_k$ units of time. Therefore, if it can be shown the condition in equation (8.7) is true over this duration, then it will always be true. This means that the condition in equation (8.9) must be shown to be true.

$$xT_k \leq t \leq xT_k + T_k, x \in \mathbb{N} : \left\lceil \frac{t}{T_{COMP}} \right\rceil \geq \sum_{j \in ST} \left| \left\lceil \frac{t - O_j}{T_k} \right\rceil \right|_0 \tag{8.9}$$

Based on the definition of the composite task's period in section 8.3.1, it can be stated that $iT_{COMP}$ is always less than the offset of the corresponding task in the set ST4 - the "corresponding task" is the $i^{th}$ member of set ST when the set ST is ordered according to increasing offset. Therefore, it can be stated that the composite task is always released before the "corresponding task" of the original tasks. Hence, if it can be shown that $N$ ($N$ being the number of tasks forming the composite task.) releases of the composite task always has a greater interference (for all time in the range $xT_k \leq t \leq xT_k + T_k$) than the tasks from

which the composite task is formed, then equation (8.9) is true. The left hand side of equation (8.9) can be expressed as shown in equation (8.10).

$$\begin{aligned}\left\lceil \frac{t}{T_{COMP}} \right\rceil &= \left| \left\lceil \frac{t}{T_k} \right\rceil \right|_0 + \left| \left\lceil \frac{t - T_{COMP}}{T_k} \right\rceil \right|_0 + \\ &\quad \left| \left\lceil \frac{t - 2T_{COMP}}{T_k} \right\rceil \right|_0 + \cdot\cdot + \\ &\quad \left| \left\lceil \frac{t - (N-1)T_{COMP}}{T_k} \right\rceil \right|_0 \end{aligned} \tag{8.10}$$

Or alternatively,

$$\left\lceil \frac{t}{T_{COMP}} \right\rceil = \sum_{i=0}^{N-1} \left| \left\lceil \frac{t - iT_{COMP}}{T_k} \right\rceil \right|_0 \tag{8.11}$$

Therefore, equation (8.9) becomes

$$xT_k \leq t \leq xT_k + T_k, x \in \mathbb{N} : \sum_{i=0}^{N-1} \left| \left\lceil \frac{t - iT_{COMP}}{T_k} \right\rceil \right|_0 \geq \sum_{j \in ST} \left| \left\lceil \frac{t - O_j}{T_k} \right\rceil \right|_0 \tag{8.12}$$

Based on the definition of the composite task in section 8.3.1, equation (8.13) can be stated. Therefore, the condition in equation (8.12) is true.

$$iT_{COMP} \leq O_j \tag{8.13}$$

Therefore, the condition in equation (8.7) is true. Hence the composite task causes greater interference than the tasks that form it.

□

**Theorem 5.** *The worst-case response time of the tasks in the task set, other than the composite task, is not lower than before the composite task is formed.*

**Proof**

An effect of the composite approach is that priorities within the system may change. This means a lower priority task that previously causes blocking on a particular task may not block, and visa versa. Theorem 4 has already shown the composite task causes greater interference than the tasks that formed it, but what about blocking? Based on the definition of the composite task in section 8.3.1, it can be stated the composite task has an equivalent or higher priority than the

tasks from which it is created. The reason is that the minimum deadline is used and priorities are assigned using DMPO. Therefore, the set of lower priority tasks with the composite approach is a superset of the lower priority tasks without the composite approach. Hence the blocking and the interference are increased.

□

### 8.3.3 Selection of Offsets

The rigorous treatment of offsets is a new area that allows scope to optimise the chosen timing constraints. At present, it is known that there is a need for offsets within the system, however there is no definition to date of what they should be. For example, consider a separation requirement specified for two tasks. As long as the minimum separation between the two tasks is maintained and the second task meets its required deadline, then the size of the offset may be increased. Therefore, it is proposed that to some extent offsets and deadlines are treated as free variables. The aim would be to choose the values of offsets to enforce the necessary requirements as well as to optimise the schedulability analysis. This is referred to as a "free variable" argument.

The approach that could be taken is to treat the tasks with offsets in a similar fashion to allocating tasks in the cyclic schedule. For N tasks (each having a period of T) that require offsets, the following offsets would be assigned to tasks:

$$0,\ \frac{T}{N},\ \frac{2T}{N},\ \cdot\cdot,\ \frac{(N-1)T}{N}$$

Based on the definition of the composite task in section 8.3.1, the offset chosen with the free variable argument results in the largest period for the composite task being obtained. Hence the likelihood of the task set being schedulable is increased.

The free variable approach could result in tasks having greater offsets than necessary, however this could provide better results. A constraint over the assignment of offsets to tasks is that the offsets should only ever be increased. In the event that the offset values provided are not sufficient (i.e. the constraint could be broken) to map the timing requirements to tasks then in effect the number of slots could be increased. Therefore, the choice of offsets is as below, where M is greater than N:

$$0,\ \frac{T}{M},\ \frac{2T}{M},\ \cdot\cdot,\ \frac{(M-1)T}{M}$$

If the task set in Table 8.1 is considered using the free variable argument with M=N=4, this would result in the tasks A, B, C and D having offsets of 0, 6250, 12500 and 18750 respectively. This would be invalid since the modified value for task C's offset would be less than the original. However, if M=5, then the choice of offsets for tasks A, B, C and D that would be valid is 0, 10000, 15000 and 20000 respectively. Using the free variable argument, the period of the composite task is equal to $\frac{T}{M}$.

In many cases, the task deadlines may also be treated in a similar manner as the offsets. Consideration of the definition of the composite task in section 8.3.1, the choice of deadline for the composite task is maximised if a task's deadline is equal to the offset of the task with the next largest offset in the transaction. For the example in Table 8.1, deadlines for task A, B, C and D should be 10000, 15000, 20000 and 25000.

## 8.4 Evidence of Effectiveness

To understand whether the composite form of analysis is effective requires comparison with the exact approach. Analysis was attempted with task set characteristics generated pseudo-randomly with a realistic range for the iteration rate [25 ms,1000 ms]. Due to the typical sizes of the least common multiple, which in this case can be as large as $1000^{\text{Number of Tasks}}$, the computational complexity did not allow any form of comparison to be carried out. Therefore, the analysis was rationalised to task set characteristics that could be feasibly expected and computed.

The task set characteristics were:

1. The iteration rates were 25 ms, 50 ms, 100 ms, 200 ms or 1000 ms.
2. The worst-case execution time of the tasks are in the range (0, 2.5] ms.
3. The offsets were assigned either randomly within the range (0,task iteration rate) or using the free variable argument.
4. The deadlines were maximised so that:
   (a) Tasks without offsets have a deadline equal to their period.
   (b) Tasks with offsets have a deadline equal to the offset of the next task in the set *ST2* defined in section 8.3.1. In the case of the task with the largest offset at a particular iteration rate (i.e. the last member of set *ST2*), the deadline is equal to the period of the task.

Figures 8.1 and 8.2 give two comparisons of the three approaches (inexact analysis in section 8.1, exact analysis in section 8.2, and composite analysis in section 8.3) discussed in this chapter. The comparisons are; without the free variable argument and with the free variable argument respectively. For the figures, the label "INEXACT" refers to analysis of task sets neglecting offsets described in section 8.1, the label "COMP" refers to the results of the composite analysis and the label "EXACT" refers to the results of the exact form of analysis. The y-axis of both graphs give an effectiveness rating (the evidence is gathered over 1000 samples per data point) that relates to the probability an approach will find a solution.

The analysis was performed for a number of task set sizes and over a range of resource bands. Each resource band covers a range of 10% processor utilisation, the lower limit of a particular resource band is labeled on the x-axis. For an example location on a line, if the x-axis has a value of 30, then the point relates to a resource band [30%,40%). The graphs have a number of lines drawn that correspond to the number of tasks within the set and the approach used, i.e. "Tasks 20 - COMP" represents composite analysis of task sets with twenty tasks. The exact approach always gives a value for effectiveness of 1, i.e. 100% of task sets are found to be schedulable. The reason is the analysis is only performed on schedulable tasks and schedulability is proven using the exact approach.

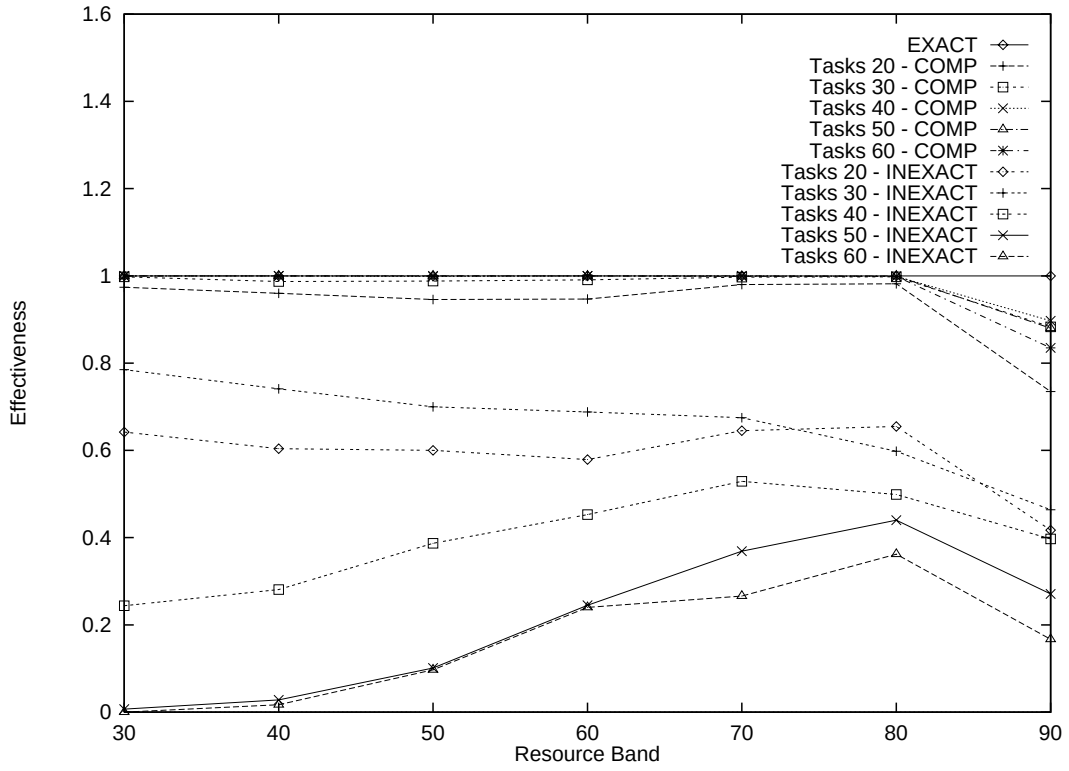


Figure 8.1: Comparison of the Approaches Without the Free Variable Argument

The results show three facts which are:

1. The free variable argument leads to an improvement in schedulability.

2. For a resource range [30%,90%) the effectiveness of the composite offset analysis is above 90%. However, the effectiveness falls at the higher levels of resource in the range [80%,100%).

3. The composite approach is significantly more effective than the inexact approach.

### 8.4.1 Effectiveness of the Restricted Computational Model

The simulation study is repeated without the improved analysis for the restricted computational model derived in sections 4.3.4 and 4.3.5. The aim is to highlight the benefits of the improved analysis. Figure 8.3 shows the effectiveness of the composite approach (labeled COMP) without the improved blocking model, compared to the composite approach with the improved blocking model (labeled

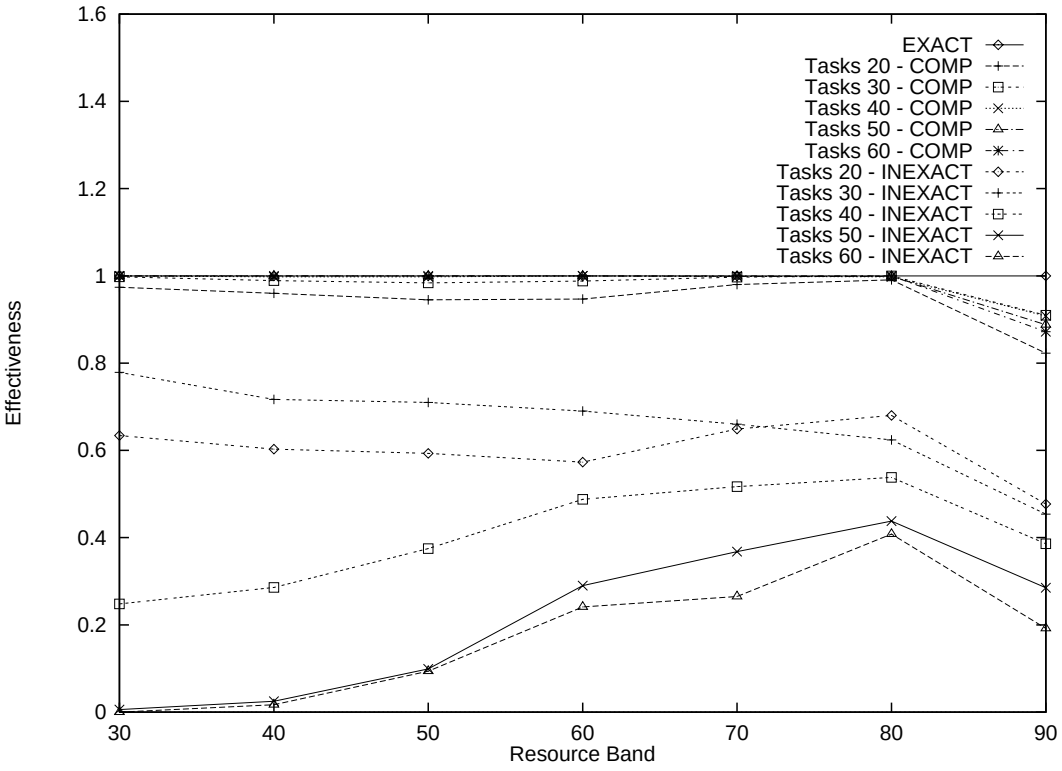


Figure 8.2: Comparison of the Approaches with the Free Variable Argument

IMPR. MODEL). The results were generated under the same conditions as those for section 8.4. The results show an improvement in the effectiveness obtained over a range of task set sizes. However, the greatest benefit is illustrated in Figure 8.4 with resource levels greater than, or equal to 90%, again over a range of task set sizes. Effectiveness is up to 20% better with the improved schedulability analysis. When the resource level is less than 90%, the impact of blocking caused by the non-preemptive scheduler tends to be less important.

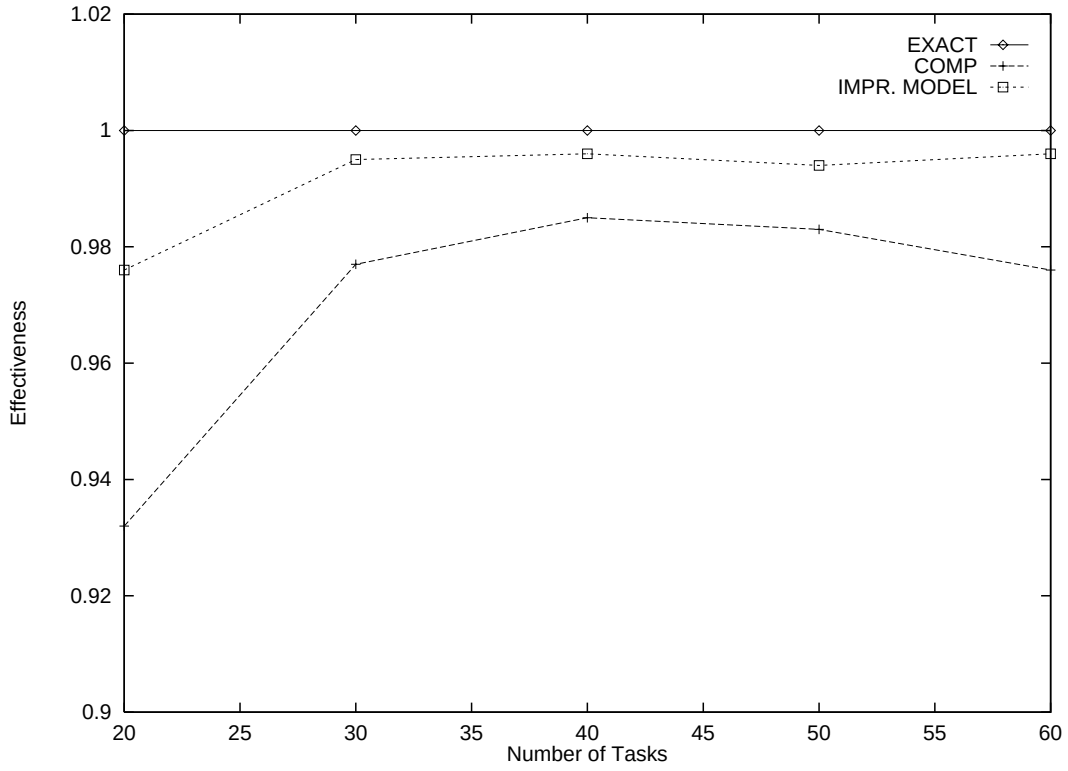


Figure 8.3: Comparison of the Composite Approach With/Without the New Blocking Model, Resource Range [30%,100%]

## 8.5 Further Improvement to the Composite Offset Analysis

This section considers the composite offset analysis, and the pessimism it introduces. There are a number of sources of pessimism within the analysis due to the less than ideal attributes ($D_{COMP}$, $T_{COMP}$, $C_{COMP}$) calculated for the composite task. The use of the free variable argument effectively improves the values of $D_{COMP}$ and $T_{COMP}$. However, $C_{COMP}$ can still be a particularly severe source of pessimism.

Control loops, where there are three tasks, provide a good example of where pessimism may arise. The first and last tasks would deal with the sensor and actuator respectively. These tasks are likely to have a comparatively small worst-case execution time. The second task would deal with data calculations and would invariably have a larger worst-case execution time than the other two tasks. According to the approach in section 8.3.1, the composite task's worst-

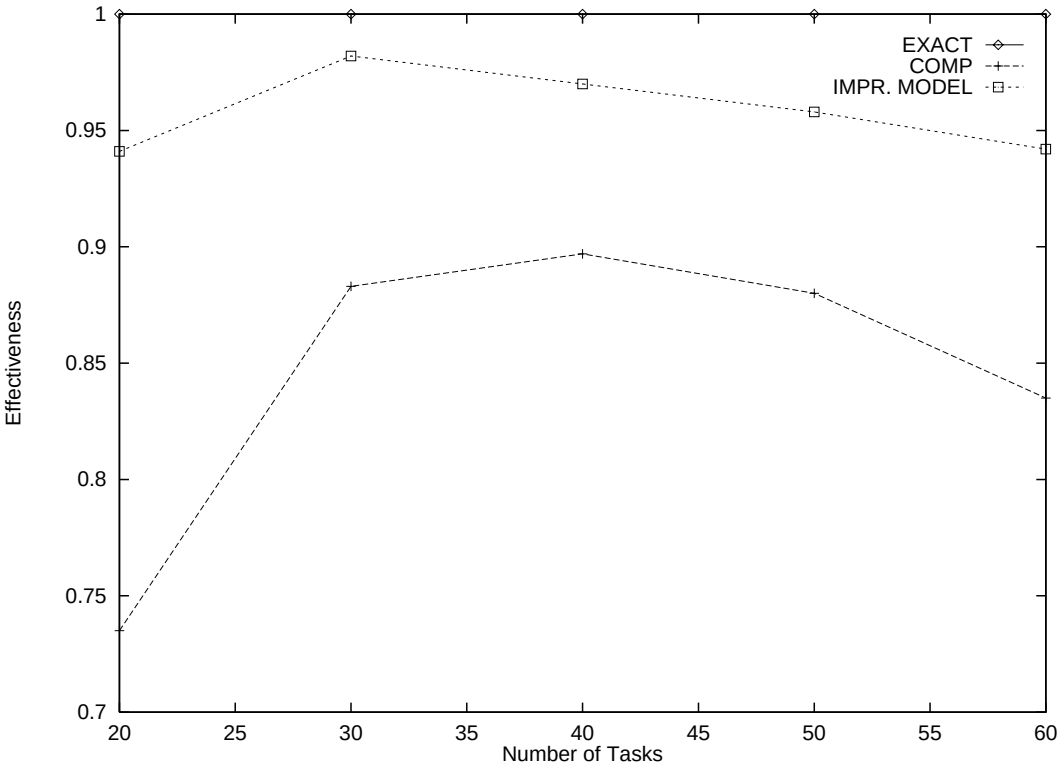


Figure 8.4: Comparison of the Composite Approach With/Without the New Blocking Model, Resource Range [90%,100%]

case execution time is equal to the worst case execution time of the second task in the transaction. When the composite task is representing the first and third tasks, there is a large amount of pessimism with the composite approach.

The amount of pessimism can be substantially reduced by varying the value of $C_{COMP}$ with time. For each composite task, a set ST4 is defined which is an ordered version of the worst-case execution times of the tasks in set ST. The set ST is defined in section 8.3.1. The ordering is in accordance with decreasing value of worst-case execution time.

*For the task set in Table 8.1, the set ST4 is defined as an ordered version of the worst case execution times of tasks A, B, C and D. Therefore, ST4 is an ordered version of* {*2000, 1500, 1500, 1500*}*. Clearly, these worst-case execution times are already in descending order as required. Hence ST4 is* {*2000, 1500, 1500, 1500*}*.*

Equation (8.14) can be used for calculating the worst-case execution time of the composite task for any response time $R_i$.

$$C_{COMP} = \frac{\sum_{\forall l: l \in ST2} \left\lceil \frac{R_i - O_l}{T_l} \right\rceil ST4(\#l)}{\left\lceil \frac{R_i}{T_{COMP}} \right\rceil} \tag{8.14}$$

where ST4 (#l) represents the $l^{th}$ member of ST4, and
#l is the index into the set ST2 in the range [1,Number of Tasks in Set ST].

Equation (8.14) provides the composite task's worst-case execution time by working out the interference of the tasks that form the composite task. The result is then divided by the maximum number of times the composite task can execute. The interference calculation is based on the tasks that form the composite task executing in the order of descending worst-case execution time. This task ordering is assumed in order to ensure the worst possible interference is obtained to guarantee the test is sufficient. Equation (8.14) represents the value of $C_{COMP}$, which is the worst-case average for the computation time of the tasks that form the composite task over the period of interest.

**Observation 7.** *The worst-case execution time of the composite task can be varied with time by using the worst-case execution times of the tasks that form the composite task cyclically in descending order.*

**Argument**

Each task that forms the composite task can only execute once in the period from the critical instant, to a point in time that is the least common multiple of the tasks' periods later (i.e. time in the range [critical instant, critical instant+period of the tasks that forms the composite task)). If the task execution order is sorted in accordance with descending worst-case execution time, then this is the worst-case situation because the greatest interference is caused to other tasks.

The tasks' worst-case execution times are not used cyclically in the order defined in set ST2 starting with the task that has the maximum worst-case execution time. The reason is worse interference could be caused dependent on the tasks' worst-case execution times that follow the task with the maximum worst-case execution time. For example, if the tasks that form the composite task have

worst-case execution times as follows: 5, 20, 1, 1, 1, 18, and 19. Then, the worst-case sequence of tasks could start with the task that has an execution time of 18 (i.e. the sequence would be 18, 19, 5, 20, 1, 1, 1) rather than the task with the maximum worst-case execution time (i.e. the sequence would be 20, 1, 1, 1, 18, 19, 5).

□

To illustrate how equation (8.14) works, an example is provided. The task set in Table 8.1 with a current value of $R_i$ of 27000. In this case the higher priority tasks (the set $j$) than the task priority analysed is the composite task. The example shows how the worst-case execution time used in the schedulability analysis equations is effectively reduced from 2000 to 1700, which is an improvement of 15%.

$$
\begin{aligned}
I_{COMP} &= \sum_{\forall l: l \in ST2} \left\lceil \frac{R_i - O_l}{T_l} \right\rceil ST4(\#l) \\
&= \sum_{\forall l: l \in \{\text{tasks A, B, C, D}\}} \left\lceil \frac{R_i - O_l}{T_l} \right\rceil ST4(\#l) \\
&= \left\lceil \frac{R_i - O_A}{T_A} \right\rceil ST4(1) + \left\lceil \frac{R_i - O_B}{T_B} \right\rceil ST4(2) + \\
&\quad \left\lceil \frac{R_i - O_C}{T_C} \right\rceil ST4(3) + \left\lceil \frac{R_i - O_D}{T_D} \right\rceil ST4(4) \\
&= \left\lceil \frac{27000 - 0}{25000} \right\rceil 2000 + \left\lceil \frac{27000 - 6250}{25000} \right\rceil 1500 + \\
&\quad \left\lceil \frac{27000 - 13000}{25000} \right\rceil 1500 + \left\lceil \frac{27000 - 18000}{25000} \right\rceil 1500 \\
C_{COMP} &= \frac{8500}{\left\lceil \frac{R_i}{T_{COMP}} \right\rceil} \\
&= \frac{8500}{\left\lceil \frac{27000}{6000} \right\rceil} \\
&= \frac{8500}{5} \\
\therefore C_{COMP} &= 1700
\end{aligned}
\tag{8.15}
$$

To show the effectiveness of the approach the same simulation conditions are used in section 8.4, except the worst-case execution times are altered. Instead the simulation is performed for transactions always having three tasks, the first

and last task having a worst-case execution time in the range [1,500] and the second task having a worst-case execution time in the range [1,5000].

Figure 8.5 shows the effectiveness of the composite approach (labeled COMP) without the improvement compared to the composite approach with the improvement (labeled IMPR. MODEL). For comparison purposes, results are provided for the analysis where offsets are ignored (labeled IGNORE OFFSETS). The two numbers (X,Y) at the end of each label (e.g. COMP_X_Y) indicate the resource range (i.e. the resource utilisation of the task sets is in the range [X%,Y%)) for the task sets.

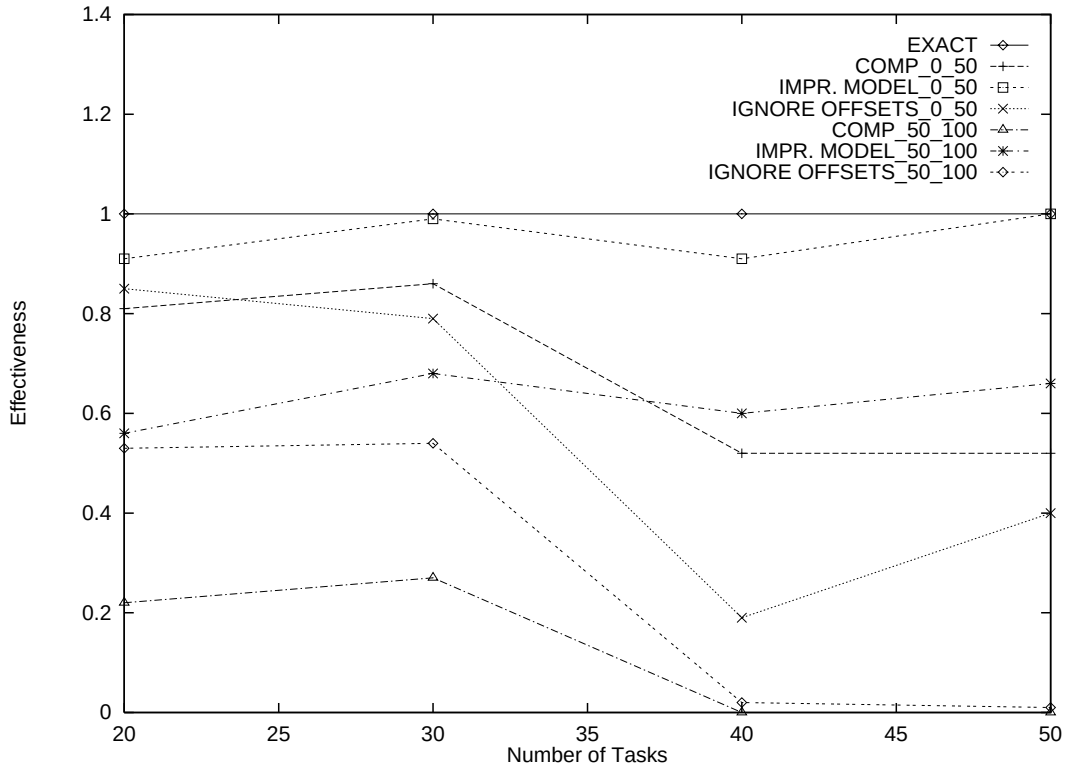


Figure 8.5: Comparison of the Approaches

Figure 8.5 shows that in this scenario the original composite approach is not always effective compared to the approach that simply ignores offsets. However, the modified composite approach performs much more effectively (up to 50% more effective) than either of the other two approaches.

## 8.6 Summary

This chapter has attempted to derive an appropriate approach to timing analysis for task sets that feature offsets. The technique developed compares favourably with the criteria for successful technology transfer.

1. *Certification* - The approach provides analysis that guarantees the system's timing behaviour. Therefore, the results of the analysis can be used as part of the certification evidence of the system.

2. *Reuse* - The "standard" schedulability analysis defined in earlier chapters is reused. The only changes necessary is a limited amount of pre- and post-processing. The processing is to establish the offsets of the tasks and messages, and then check for convergence.

3. *Sufficiency* - The approach does not place restrictions on the computational model. Therefore, the sufficiency criterion is satisfied. The reduced pessimism compared to the inexact approaches increases the likelihood that the timing requirements are met. It should be noted that the appropriate selection of offset requirements increases the likelihood of the system being schedulable.

4. *Understanding* - The approach is considered understandable as demonstrated by the fact it has been technology transferred to Rolls-Royce [96] and the Guards project. The GUARDS project is an ESPRIT funded project that addresses the development of architectures, methods, techniques, and tools to support the design, implementation and validation of critical real-time systems.

An additional benefit is computational complexity is reduced compared to exact analysis.

# Chapter 9

# Transition from Uniprocessor to Distributed System

Work on the hardware aspects of the system has shown the benefit (a saving in cost and weight) of a transition from a uniprocessor to a distributed system. This transition can reduce the amount of cabling between the processor units and sensors/actuators [5]. Another advantage of the transition is that there should be an increase in the available processing resource within the system.

The production and verification of distributed real-time systems is very complex, particularly for safety critical systems with the need to certify the product. There are a number of technical issues associated with distributed systems. These issues include: allocation [106], task attribute assignment [107], increasing robustness to change [64], and timing analysis [108]. This chapter concentrates on the last three of these issues, making the assumption that the allocation of functionality to processors is largely based on the physical position of devices. The real challenge with distributed scheduling is the efficient implementation and verification of transactions involving more than one processor. This is because of the conflicting requirements imposed on the different components.

The work in this chapter is intended to build on the work performed for uniprocessor systems in Chapters 3, 4, 5 and 6 so that a smooth transition can be attained. The benefits of this strategy are the ability to reuse the investment in existing tools and the education of staff as well as providing the ability to break the system up into smaller parts.

The structure of the chapter is as follows. Section 9.1 shows how the composite offset analysis of Chapter 8 may be used as part of a distributed approach. Section 9.2 presents an example of how the distributed analysis based on composite offset analysis may be applied to a system's timing requirements, following a transition from a uniprocessor to a distributed system. Finally, section 9.3 presents evidence of effectiveness for the composite offset analysis compared to the Phase Modification Protocol analysis and the release jitter based approaches described in section 3.2.

## 9.1 The Composite Offset Analysis

Section 3.2 of the literature survey provides background on previous approaches (Release Jitter and Phase Modification Protocol) that have been developed for the problem of distributed scheduling and timing analysis. Section 3.2 highlights a number of problems. In brief, the problems are:

1. the release jitter approach is based on an event-driven system, which is considered difficult to certify and the pessimism is too great,
2. the exact analysis has high computational complexity, and
3. for both approaches is a lack of robustness to change, i.e. a change on one processor has a system-wide impact.

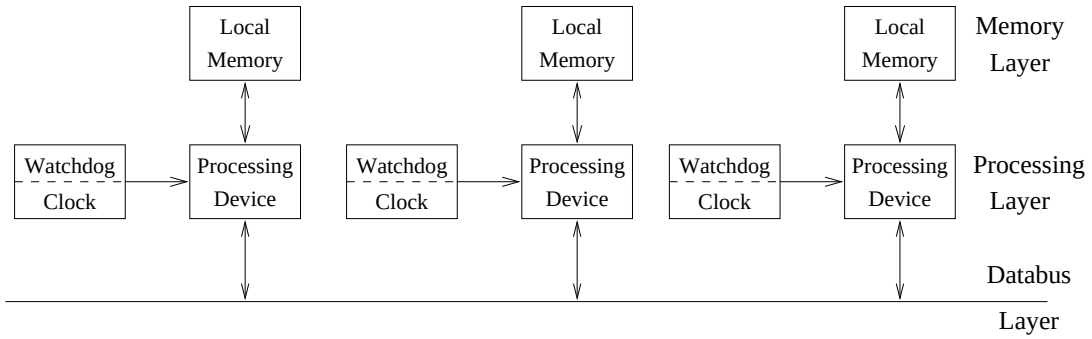


Figure 9.1: Basic Architectural Structure

The system architecture being considered is general. The architecture in Figure 9.1 illustrates the main points, which are:

1. The processors may be physically separated by a significant distance. Therefore it is assumed communication between processors is performed using a databus rather than via a back-plane.

2. All processors communicate via a fully interconnected fixed priority databus, i.e. messages are routed between processors via one bus without the need for gateways and multiple-hops.

3. The architecture is based on multiple versions of the infrastructure (i.e. processor, timing watchdog, real-time clock and local memory) discussed in chapters 4, 5 and 6.

4. It is assumed a mechanism for providing a global-time base is available.

The aim of this section is to develop an approach to scheduling and timing analysis based on the composite offset analysis developed in Chapter 8.

### 9.1.1 Computational Model

The computational model developed is intended to address many of the drawbacks of the release jitter and exact analysis, whilst harnessing the advantages. The computational model is proposed with the following goals in mind:

1. increase the robustness to change,
2. sporadic tasks are not utilised,
3. existing uniprocessor scheduling techniques from Chapters 4-8 can be reused,
4. low computational complexity, and
5. less pessimism than the release jitter approach.

The implementation aspects of our approach are the same as the Phase Modification Protocol discussed in section 3.2.2 and [73], i.e. offsets are used to enforce precedence. In brief, a task in the transaction is given an offset that is greater than or equal to the response time of the preceding task (or message that delivers data from the preceding task in the case where the tasks exist on different processors) in the transaction. This strategy achieves precedence between tasks irrespective of whether the tasks exist on the same processor or different processors. However, in an implementation model based on offsets a global time base is necessary.

Figure 9.2 shows an execution sequence of a number of tasks that form a transaction across a distributed system, and the messages that communicate data between tasks. Figure 9.2 can be used to show how the time during which a task or message may execute is controlled. For example, in the case of task *t2* the duration of allowed execution commences when message *m1* has definitely arrived, i.e. $O_{t2} = R_{m1}$. Message *m2* is then scheduled for when task *t2* has completed, i.e. $O_{m2} = R_{t2}$, and so on. By giving a task an offset such that its dispatch (or release for simplicity) is always greater than the worst-case response time of the event trigger (i.e. the worst-case arrival time of the message), precedence is maintained even across a distributed system.

Controlling precedence using offsets means that timing analysis is required for task sets where tasks have offsets. It is proposed that the timing analysis developed in Chapter 8 is used.

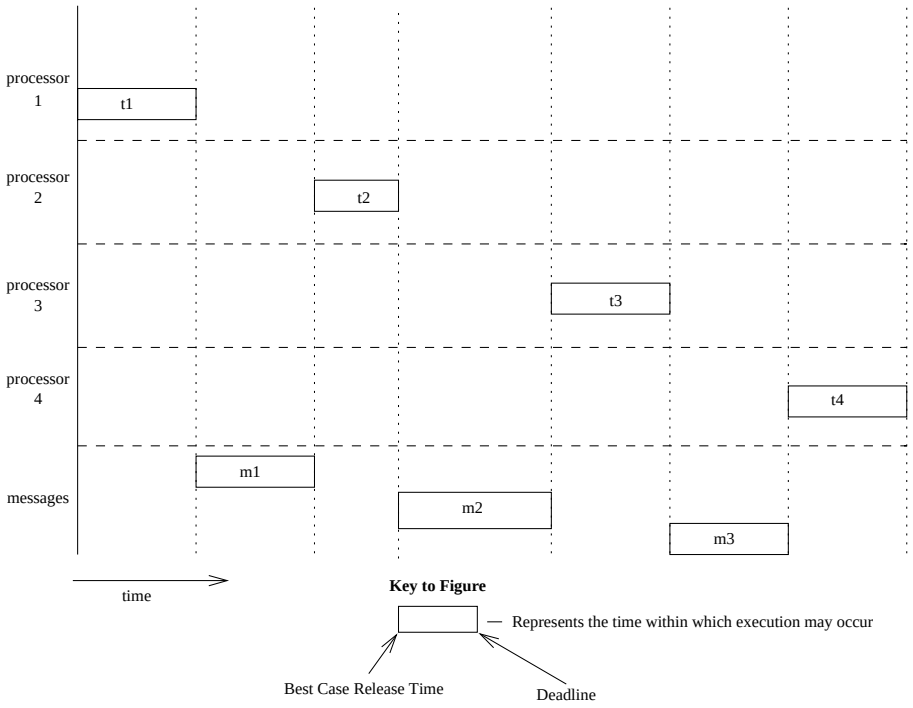


Figure 9.2: Diagram to Illustrate the Timing Analysis of a Transaction

### 9.1.2 Use of The Free Variable Argument

Chapter 8 highlights potential pessimism in the composite analysis when the value of the tasks' offsets are small. The definition of a composite task in section 8.3.1 means that if one of the values for the offsets is small, then the period of the composite task is also small. This causes problems when schedulability analysis is performed because of the interference term in equation (4.12). The reason is that the composite task would have a higher priority and a smaller period, which causes more interference.

A potential solution is seen as the free variable argument introduced in section 8.3.3. The approach taken is to split the offset tasks up in a similar fashion to allocating tasks in the cyclic scheduler. The $N$ tasks in the transaction (each having a period of T) are assigned the offsets given below so that the tasks execute in the required order:

$$0,\ \frac{TD}{N},\ \frac{2TD}{N},\ ..,\ \frac{(N-1)TD}{N}.$$

For example, the *Mth* task has an offset of $\frac{M-1}{N}TD$, where $TD$ is the transaction's end-to-end deadline. In effect adopting this approach means that the free variable

argument is used at two levels, across the system for distributed transactions and on the individual processor.

The following equation is suggested for applying the free variable argument:

$$\textbf{slot time} \;=\; \frac{t}{\text{no. of tasks in the transaction}} TD \tag{9.1}$$

where $t$ is an index to the $t^{th}$ task in the transaction

The value of *slot time* is calculated with equation (9.1) represents the start time of a slot allocated to a particular task within a transaction, with each task having its own slot. The slots are ordered in accordance with precedence over the available response time of the transaction. The free variable argument is applied as shown in Algorithm 5 so that an offset is assigned that ensures the task executes after its allocated slot time and the preceding task/message has completed. Algorithm 5 satisfies the constraint that offsets are never decreased.

**Algorithm 5.** - Algorithm for determining a task's offsets

*if (the WCRT of the preceding task/message < slot time)*
    *offset = slot time*
*else*
    *offset = WCRT of the preceding task/message*

An observation concerning the use of offsets and the associated slot times is the analogy with the TDMA (Time Division Multiple Access) communications model. TDMA is a commonly used communications model, where each message is allocated a fixed slot in a round robin scheduler. ARINC 629 [12] represents an example of a communications model that can provide TDMA.

The difference between the model defined in this chapter and the TDMA model is that whilst messages are assigned slots, many messages may have the same or overlapping slots. In these cases, arbitration is provided by the priorities. Therefore, the problems of maintainability that are traditionally associated with the cyclic scheduler do not arise. The approach in this section could equally be applied to a system where tasks are scheduled with the fixed priority technique, and messages using the TDMA approach. There are two benefits associated with

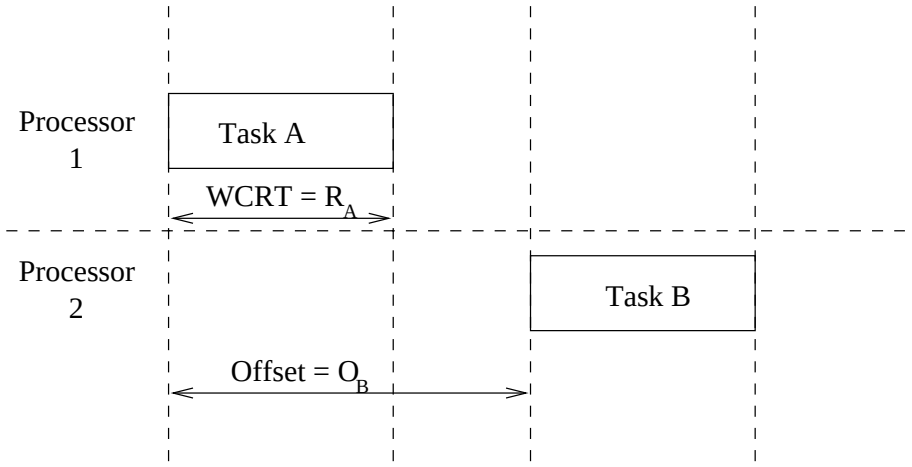


Figure 9.3: Phasing of Tasks on Different Processors

this observation. These are: the evolution of technology has taken a smaller step resulting in a less steep learning curve, and there is a high probability that certification evidence may be reused. Again, this helps increase the likelihood of acceptance during the technology transfer exercise.

### 9.1.3 Robustness to Change

One of the perceived benefits of the free variable argument is the robustness to change, which enhances maintainability by saving verification time. The reason is within defined bounds a change on one processor does not necessitate a system wide re-verification.

Figure 9.3 illustrates how the timing characteristics of processor 1 may be modified until $R_A \geq O_B$, without affecting the scheduling of processor 2, since $D_A = O_B$. Therefore, if the software on one processor is modified, then only that processor needs to be re-analysed as long as the timing requirements are met. When the timing requirements are no longer met, the timing analysis and task attribute assignment for the system is repeated. Therefore, the approach represents a partial move towards the integration of task attribute assignment and timing analysis, which helps meet our reduction in pessimism and robust analysis objectives.

In contrast, both the release jitter and exact approaches have the release time of a task in the transaction as always being equal to the preceding task's completion

time. This means any change could have system wide repercussions. It should be noted that the cost of the occasional system wide change is comparable to the release jitter approach after any change, (and significantly less than the exact approach) since the steps that need to be taken are the same.

There are a number of envisaged benefits of the free variable argument, including: resources are spread through time, and the value of each offset can never be too small. For the purpose of this work, the free variable argument is applied for analysis reasons only. However, real gains may be achieved by actually manipulating the requirements to achieve lower pessimism and better scalability. The advantages of the composite offset approach are listed at the beginning of this section. However, there is the obvious disadvantage that some pessimism still exists.

## 9.2 Explanation of the Distributed Systems Timing Analysis Using the Composite Offset Analysis

The purpose of this section is to present an example set of system timing requirements so that the effect of a transition from a uniprocessor to a distributed system may be understood with respect to the timing analysis approach - release jitter, exact analysis and composite offset.

### 9.2.1 System Characteristics

The set of requirements to be used as an example is illustrated by the task set in Table 9.1 and the transactions in Table 9.2.

The columns of Table 9.2 represent:

$Id$ is the name given to tasks and transactions
$C$ is the worst-case execution time of the task
$T$ is the period of the task
$D$ is the deadline of the task after the transition

| Id | T | D | C |
|---|---|---|---|
| 00 | 725000 | 725000 | 3896 |
| 01 | 25000 | 25000 | 3964 |
| 02 | 325000 | 325000 | 878 |
| 03 | 25000 | 25000 | 1378 |
| 10 | 725000 | 725000 | 2228 |
| 11 | 25000 | 25000 | 3612 |
| 12 | 225000 | 225000 | 1230 |
| 13 | 900000 | 900000 | 1232 |
| 20 | 725000 | 725000 | 1668 |
| 21 | 25000 | 25000 | 4672 |
| 22 | 950000 | 950000 | 4784 |
| 23 | 400000 | 400000 | 4696 |

Table 9.1: Task Set's Characteristics

| Transaction Id | 1st Task | 2nd Task | 3rd Task | Transaction Deadline |
|---|---|---|---|---|
| A | 21 | 11 | 01 | 25000 |
| B | 00 | 20 | 10 | 725000 |

Table 9.2: Transaction Characteristics

Tables 9.1 and 9.2 provide both the system's timing characteristics. The system's timing characteristics are defined in columns Id, T, D and C of Table 9.1. In addition, the system's timing requirements includes two transactions given in Table 9.2. Each transaction is given an identifier, a precedence order (i.e. 1st task followed by 2nd task followed by 3rd task), and an end-to-end deadline requirement. For example, transaction A refers to a transaction requirement that task 21 is followed by task 11 followed by Task 01 that should be complete within a time of 25000.

Table 9.3 contains the results of the schedulability analysis for the single processor case. Column $R_S$ is the worst-case response time for the single processor case. The results were obtained assuming attributes are assigned using the approach in Chapter 6. The results in column $R_S$ indicate that the timing requirements are met since $\forall \text{tasks} : R_S \leq D$.

| Id | T | D | C | $R_S$ |
|---|---|---|---|---|
| 00 | 725000 | 725000 | 3896 | 24326 |
| 01 | 25000 | 25000 | 3964 | 8809 |
| 02 | 325000 | 325000 | 878 | 15734 |
| 03 | 25000 | 25000 | 1378 | 10187 |
| 10 | 725000 | 725000 | 2228 | 26554 |
| 11 | 25000 | 25000 | 3612 | 13799 |
| 12 | 225000 | 225000 | 1230 | 14856 |
| 13 | 900000 | 900000 | 1232 | 46257 |
| 20 | 725000 | 725000 | 1668 | 45025 |
| 21 | 25000 | 25000 | 4672 | 18471 |
| 22 | 950000 | 950000 | 4784 | 51041 |
| 23 | 400000 | 400000 | 4696 | 20430 |

Table 9.3: Task Set's Characteristics and Schedulability Analysis Results

### 9.2.2 Making the Transition to a Distributed System

A simple architecture is given in Figure 9.1 that provides the basic architecture required to support the transition. Table 9.4 and Table 9.5 provides the scheduling information and results for a transition from a uniprocessor to a distributed system. The architecture features three processors that are fully interconnected by a common databus.

Table 9.4 provides additional scheduling information to support this transition. Column $N$ identifies the processor to which each task has been allocated after the migration to a distributed system. In column $N$, the symbol $M$ signifies a message on the databus. The identifiers for the messages are also prefixed with an $M$. The additional schedulability analysis results columns of Table 9.4 over Table 9.3 represent:

$R_J$ is the worst-case response time if the tasks are executed on 3 processors and analysed using the release jitter approach

$R_E$ is the worst-case response time if the tasks are executed on 3 processors and analysed using the exact analysis approach

$R_C$ is the worst-case response time if the tasks are executed on 3 processors and analysed using the composite offset approach

| Id | N | T | D | C | $R_S$ | $R_J$ | $R_E$ | $R_C$ |
|---|---|---|---|---|---|---|---|---|
| 00 | 0 | 725000 | 725000 | 3896 | 24326 | 10116 | 8738 | 8738 |
| 01 | 0 | 25000 | 25000 | 3964 | 8809 | 25189 | 13139 | 24831 |
| 02 | 0 | 325000 | 325000 | 878 | 15734 | 14080 | 4842 | 8738 |
| 03 | 0 | 25000 | 25000 | 1378 | 10187 | 13202 | 1378 | 7860 |
| 10 | 1 | 725000 | 725000 | 2228 | 26554 | 28524 | 21559 | 34343 |
| 11 | 1 | 25000 | 25000 | 3612 | 13799 | 16582 | 8762 | 16169 |
| 12 | 1 | 225000 | 225000 | 1230 | 14856 | 7070 | 1230 | 7070 |
| 13 | 1 | 900000 | 900000 | 1232 | 46257 | 8302 | 2462 | 8302 |
| 20 | 2 | 725000 | 725000 | 1668 | 45025 | 30455 | 18997 | 25229 |
| 21 | 2 | 25000 | 25000 | 4672 | 18471 | 9517 | 4672 | 9517 |
| 22 | 2 | 950000 | 950000 | 4784 | 51041 | 18997 | 14152 | 18997 |
| 23 | 2 | 400000 | 400000 | 4696 | 20430 | 14213 | 9368 | 18997 |
| M10 | M | 725000 | 725000 | 117 | N/A | 11458 | 10233 | 10928 |
| M11 | M | 25000 | 25000 | 478 | N/A | 10742 | 5150 | 16981 |
| M20 | M | 725000 | 725000 | 334 | N/A | 31680 | 19331 | 26041 |
| M21 | M | 25000 | 25000 | 413 | N/A | 17329 | 9175 | 10329 |

Table 9.4: Task Set's Characteristics and Schedulability Analysis Results

Table 9.5 provides information on how the transaction is implemented on the distributed architecture. Whereas Table 9.2 provided the precedence constraints for just tasks, Table 9.5 also includes the relevant messages for transferring data. For example, message M11 carries data between task 21 and task 11, and message M21 carries data between task 11 and task 01.

When, the schedulability analysis is performed using the exact analysis approach, the results show that after the transition to a distributed architecture the system's timing requirements are still schedulable.

| Transaction Id | 1st Task | 1st Message | 2nd Task | 2nd Message | 3rd Task | Transaction Deadline |
|---|---|---|---|---|---|---|
| A | 21 | M11 | 11 | M21 | 01 | 25000 |
| B | 00 | M10 | 20 | M20 | 10 | 725000 |

Table 9.5: Transaction Characteristics

Next, the release jitter approach is used to analyse the system, the results are given in column $R_J$. This time the results indicate that task *01* of the system is no longer schedulable. The analysis is terminated as soon as this task failed its requirements, i.e. its response time exceeds its deadline. Therefore, the worst-case response times for the other tasks are not their *final* values. The final values would almost certainly be larger, and more tasks may miss their deadline. The results of the exact analysis clearly show that the release jitter approach has a great deal of pessimism. In fact, the results imply that splitting the functionality over a number of processors instead of one can reduce the likelihood of the system being schedulable. Increasing the system's resources and timing margin suggests that the system should become more schedulable. This assumes the databus does not cause too great an overhead, which in this case it doesn't.

With the composite approach, the results (given in column $R_C$) are pessimistic in comparison to the exact approach, however the system is still schedulable, i.e. $\forall$tasks : $R_C \leq D$. The example shows that the composite approach provides a useful alternative to the exact approach in cases where the computational complexity of the exact approach is prohibitive and pessimism is less of an issue.

### 9.2.3 Explanation of the Composite Approach

Table 9.6 helps to explain how the composite approach works. The columns of Table 9.6 have the same meaning as for Table 9.4. There are additional columns for; $R$ refers to the worst-case response time of each task calculated by the composite free variable approach, $P$ refers to the priority of the task/message and *Met?* refers to whether the task/message meets its deadline, i.e. $\forall$tasks:, $R_C \leq D$.

In Table 9.6 there are a number of tasks whose *Id* is of the form CXXXXX, which refers to a composite task representing tasks with period XXXXX. For instance in the case of the task C25000 on node 0, this task represents the original tasks *01* and *03* from Table 9.4. Task *01* inherits a non-zero offset from message *M21* because task *01* follows message *M21* in transaction A. Task *03* maintains a zero offset because it is not part of a transaction. Table 9.7 shows the task set's characteristics that lead to the results in Table 9.4.

| Id | N | T | C | D | R | P | Met? |
|---|---|---|---|---|---|---|---|
| C25000 | 0 | 12500 | 3964 | 8019 | 7860 | 1 | Y |
| 02 | 0 | 325000 | 878 | 325000 | 8738 | 2 | Y |
| 00 | 0 | 725000 | 3896 | 725000 | 8738 | 3 | Y |
| C25000 | 1 | 12500 | 3612 | 12500 | 5840 | 1 | Y |
| 12 | 1 | 225000 | 1230 | 225000 | 7070 | 2 | Y |
| C725000 | 1 | 362500 | 2228 | 362500 | 8302 | 3 | Y |
| 13 | 1 | 900000 | 1232 | 900000 | 8302 | 4 | Y |
| 21 | 2 | 25000 | 4672 | 25000 | 9517 | 1 | Y |
| C725000 | 2 | 362500 | 4845 | 362500 | 14301 | 2 | Y |
| 23 | 2 | 400000 | 4696 | 400000 | 18997 | 3 | Y |
| 22 | 2 | 950000 | 4784 | 950000 | 18997 | 4 | Y |
| C25000 | M | 8333 | 478 | 8333 | 812 | 1 | Y |
| C725000 | M | 241666 | 334 | 241666 | 812 | 2 | Y |

Table 9.6: Schedulability Analysis Results with the Composite Approach

There are a number of steps between Table 9.4 from Table 9.7, these are illustrated below:

1. Task *01* has an offset of 16981, which is equivalent to the worst-case response time of message *11*.

2. Task *01* and *03* are combined into a composite task with characteristics;
   $T = \min\{\frac{O_{01}}{1}, \frac{T_{03}}{2}\} = \min\{16981, \frac{25000}{2}\} = 12500$
   $D = \min\{(D_{01} - O_{01}), (D_{03} - O_{03})\} = \min\{25000 - 16981), (25000 - 0)\} = 8019$
   $C = \max\{C_{01}, C_{03}\} = 3964$

3. Schedulability analysis is performed to calculate the response times.

4. The worst-case response time of task *01* is calculated using equation (8.5),
   $R_{01} = O_{01} + R_{C25000} = 16981 + 7860 = 24841$

5. The worst-case response time of task *03* is calculated using equation (8.5),
   $R_{03} = O_{03} + R_{C25000} = 0 + 7680 = 7860$

| Id | N | T | D | C | O | R |
|---|---|---|---|---|---|---|
| 00 | 0 | 725000 | 725000 | 3896 | 0 | 8738 |
| 01 | 0 | 25000 | 25000 | 3964 | 16981 | 24831 |
| 02 | 0 | 325000 | 325000 | 878 | 0 | 8738 |
| 03 | 0 | 25000 | 25000 | 1378 | 0 | 7860 |
| 10 | 1 | 725000 | 725000 | 2228 | 26041 | 34343 |
| 11 | 1 | 25000 | 25000 | 3612 | 10329 | 16169 |
| 12 | 1 | 225000 | 225000 | 1230 | 0 | 7070 |
| 13 | 1 | 900000 | 900000 | 1232 | 0 | 8302 |
| 20 | 2 | 725000 | 725000 | 1668 | 10928 | 25229 |
| 21 | 2 | 25000 | 25000 | 4672 | 0 | 9517 |
| 22 | 2 | 950000 | 950000 | 4784 | 0 | 18997 |
| 23 | 2 | 400000 | 400000 | 4696 | 0 | 18997 |
| M10 | M | 725000 | 725000 | 117 | 10116 | 10928 |
| M11 | M | 25000 | 25000 | 478 | 9517 | 16981 |
| M20 | M | 725000 | 725000 | 334 | 25229 | 26041 |
| M21 | M | 25000 | 25000 | 413 | 16169 | 10329 |

Table 9.7: Results Generated Using the Composite Approach

The use of the free variable argument can be demonstrated by considering task *20* that forms task *C725000* using the composite approach. The offset of task *20* is 10928. Without the free variable argument the period of the composite task would be 10928, and with the free variable argument it is 362500. The resulting periods illustrate the benefit of the free variable argument that the interference on other tasks is considerably reduced. The interference is reduced because the composite task's period and deadline is increased, leading to a reduced priority and an increased chance of schedulability.

## 9.3 Simulation to Demonstrate the Relative Effectiveness of the Different Computational Models

Simulation studies are an effective way to understand the behaviour of large complex systems as demonstrated in Chapter 8. This section presents a simulation based investigation considered appropriate and realistic, and the results obtained.

Simulations have been performed with purely pseudo-random task set characteristics with a realistic range for the iteration rate of [25, 1000]. Due to the typical sizes of the least common multiple (which can be as large as $1000^{\text{Number of Tasks}}$), the computational complexity did not allow any form of comparison to be carried out. Therefore, the analysis was rationalised to task set characteristics that could be expected and feasibly computed.

The system characteristics were:

1. The iteration rates were 25, 50, 75, 100, 125, .., 1000.
2. The worst-case execution time of tasks are in the range (0,2.5].
3. The worst-case communication time of messages are in the range (0,0.25].
4. The number of nodes is in the range of [2,6].
5. The number of tasks ($N$) is in the range [10,30].
6. A number of transactions in the range [1,$N$]. A transactions has a number of tasks, equal to the number of nodes, each task executing on a different processor. The transaction deadline is equal to its period.
7. A value for effectiveness is produced over a 1000 samples. Effectiveness is the percentage of task sets calculated as schedulable, compared to exact analysis.

The simulation results for 10, 20 and 30 tasks per processor are presented in the graphs contained in Figures 9.4, 9.5 and 9.6 respectively. For each analysis

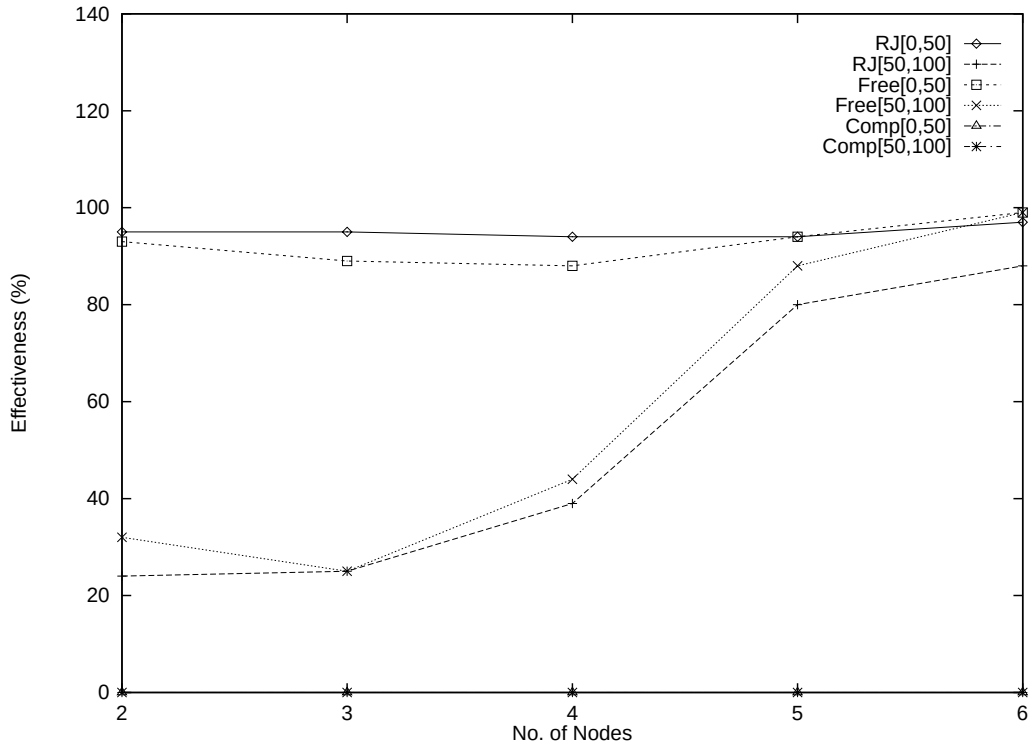


Figure 9.4: Comparison of the Simulation Results for 10 Tasks

technique there are two lines, one of the lines is the results for resources in the range [0%,50%] and the other the results for resources in the range [50%,100%]. The results clearly show the straight forward composite approach is the worst because virtually all the task sets were found to be unschedulable. The straight forward composite approach relates to the composite approach without the free variable argument being applied.

When the free variable argument is applied then there is generally less pessimism than the release jitter approach. In particular, when the resource level is higher (i.e. in the range [50%,100%]) the composite offset analysis with free variable argument performs up to 10% better than the non-free variable argument. The improvement provided with the composite offset approach is significant on top of the other benefits. Therefore, the composite offset analysis with free variable argument is viewed as a viable solution for distributed timing analysis.

The release jitter approach only performs better when the resource level is small. The better performance is a consequence of the tasks' response times being smaller, which leads to the offset or release jitter (the relevant one is depen-

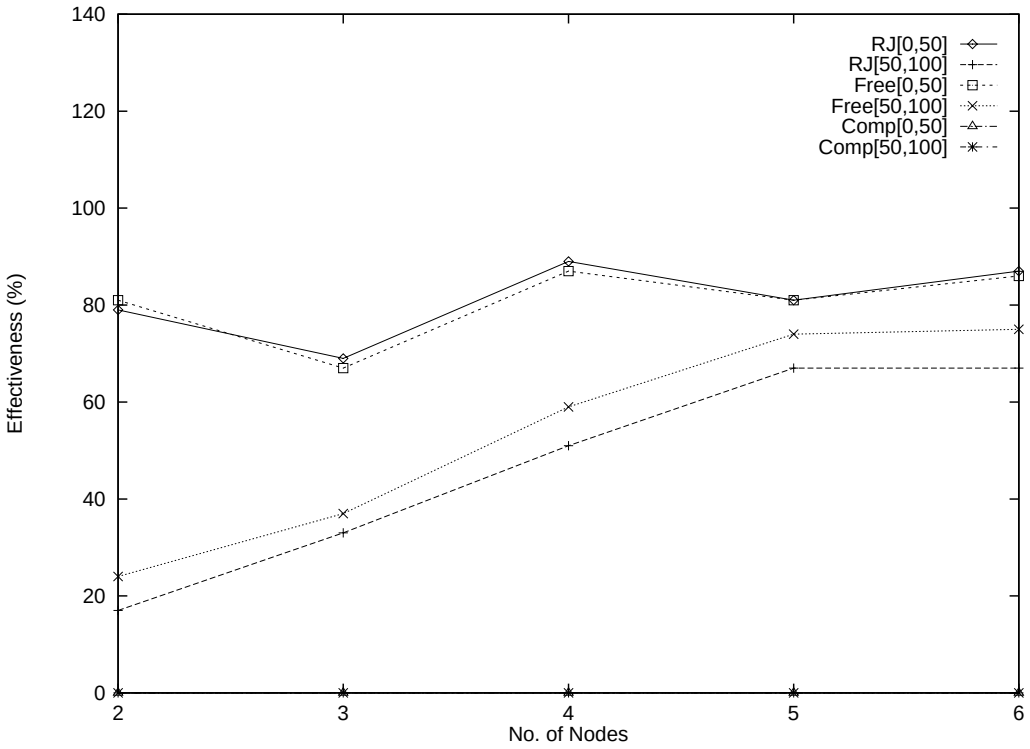


Figure 9.5: Comparison of the Simulation Results for 20 Tasks

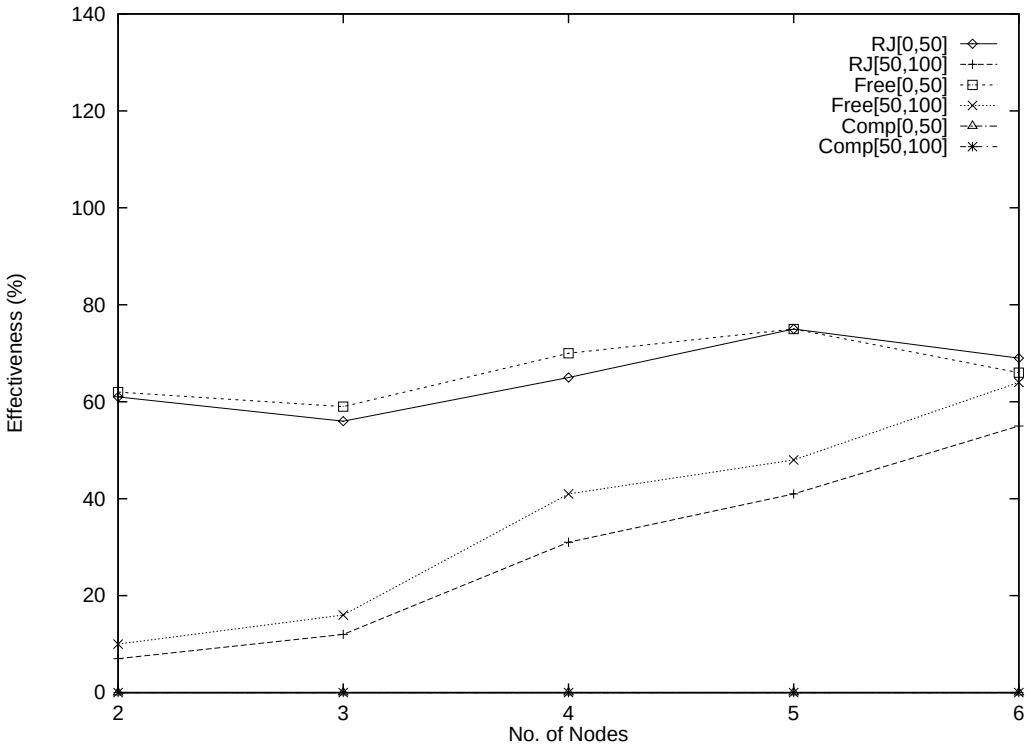


Figure 9.6: Comparison of the Simulation Results for 30 Tasks

dent on the technique being used to enforce precedence) being small. When the offset or release release jitter is small, the release jitter analysis has less pessimism than the composite offset analysis.

## 9.4 Summary

This chapter has attempted to achieve a number of goals, which were; to investigate the transition from uniprocessor to distributed systems timing, and how to reduce pessimism by combining task attribute assignment and timing analysis without significantly increasing computational complexity.

The chapter shows how the existing uniprocessor timing analysis may be used for distributed systems. However, the previous approaches based on sporadic tasks introduce too much pessimism and may be difficult to certify. An example is presented that represents a transition from executing the functionality on a single processor to executing the functionality on three processors. Each of the three processors is equivalent to the original single processor. The three processors are fully interconnected via a databus. The set is schedulable on a single processor system, but not on a three processor system analysed using the release jitter approach. This shows how making the transition to a distributed system does not necessarily make the system more schedulable even though the processor resource available is increased.

A new approach is derived using offsets to realistically spread the resource usage through time and to help remove pessimism caused by unnecessary clashes. To solve the computational complexity issues usually found with offset analysis, the composite offset analysis from Chapter 8 is used. However, with the composite approach there is still a large amount of pessimism. Therefore, a free variable argument is applied to the selection of offsets in order to better integrate task attribute assignment and the timing analysis. The combination of using offsets to model system interaction, and a composite analysis achieved significantly better effectiveness than other tractable approaches. Further benefits include; the free variable argument gives properties that are non-holistic within defined bounds, and the use of offsets is analogous to a TDMA based system, which should aid reuse.

The technique developed compares favourably with the criteria for successful technology transfer.

1. *Certification* - The approach provides analysis that guarantees the system's timing behaviour. Therefore, the results of the analysis can be used as part of the certification evidence of the system.

2. *Reuse* - The "standard" schedulability analysis defined in earlier chapters is reused. The only changes necessary is a limited amount of pre- and post-processing. The processing is to establish the offsets of the tasks and messages, and then check for convergence.

3. *Understanding* - The approach is considered understandable as demonstrated by the example. In addition, the approach has been accepted by Rolls-Royce and is being considered for use on future projects [96].

4. *Sufficiency* - The approach does not place restrictions on the computational model, and it provides a lower level of pessimism than the release jitter approach. Therefore, it can be argued the sufficiency criterion is satisfied.

# Chapter 10

# Conclusions and Further Work

The purpose of this chapter is to summarise and evaluate the work of this thesis and its contributions, as well as providing insight into further work that could be performed. The principal aims of the thesis are to augment current work on scheduling and timing analysis so that it is valid for use in real industrial safety critical hard real-time systems. In particular, two transitions are supported. These are; from cyclic scheduling to fixed priority scheduling, and from uniprocessor to distributed systems.

The first part of the thesis is to establish a base-line for the work to take place. This involved establishing the characteristics of current systems and how they are developed, which is dealt with in Chapter 2. Also, a survey was performed to examine the relevance of existing work, the results of which are in Chapter 3.

To support the first transition, an investigation is required of how existing infrastructures can be reused effectively, appropriate timing analysis derived and task attributes assigned. To support the second transition, an approach for handling distributed transactions is sought that takes advantage of the work performed for the first transition. To guide the investigation that is undertaken, four criteria for successful technology transfer are defined; certification, understanding, sufficiency and reuse. The criteria are used to judge whether proposed solutions are successful.

Chapter 4 of the work contrasts the requirements of the system to be developed with the existing theory available to determine where attention is needed. A number of areas of work are identified. As part of Chapter 4, a scheduling

approach based on only periodic tasks executing non-preemptively is chosen. This approach maximises the reuse of the existing system and documentation, which reduces the cost of change and eases the future certification. The work revisits the existing schedulability analysis and provides significant reductions in the pessimism contained in the analysis.

Chapter 5 of the work presents a way of implementing task scheduling that takes advantage of two existing mechanisms, which are; tick driven and time driven scheduling. The "hybrid" approach is intended to provide the best compromise between reuse and sufficiency. The technique uses the clock tick from the cyclic scheduler to release tasks in cases when the task's period is an integer multiple of the clock tick rate. Remaining tasks are then released in a time driven manner. The approach is novel even though it is based on existing work. The advantages are that; the minimum possible change is made to the way tasks are released, no restrictions are placed on the timing requirements, and tasks are released with no jitter. Having no release jitter increases the likelihood of meeting the timing requirements. The contribution of this chapter is a philosophy for producing an infrastructure that is tailored to the requirements whilst recognising the importance of reuse. The approach provides the best possibility of achieving the system's timing requirements at the same time as minimising the changes to the system.

Chapter 6 presents an algorithm for assigning attributes to tasks so that timing requirements are met. The technique is intended for requirements existing on a single processor system. The requirements that are satisfied (where possible) by the task attribute assignment process are; task's jitter, task's separation and transaction's deadline. The task attributes manipulated are offsets and deadlines. The approach is a novel contribution for a number of reasons. These are; the resulting task attributes make it relatively easy to determine by inspection that the timing requirements are met, and the assignment preserves the attributes imposed by other constraints. The latter point is considered the most important. Most existing approaches to task attribute assignment suffer from the problem that all attributes are considered to be flexible. However, experience has shown that tasks' periods should not be manipulated and tasks' deadlines should never be increased. Otherwise, existing constraints that have previously been satisfied may be broken. Most existing techniques do not place these restrictions on the

task attribute assignment, and hence are deemed unsuitable.

Chapter 7 uses a realistic case study of an aircraft engine control system to prove the techniques developed in Chapters 4 - 6 are sufficient. As part of the evaluation, a scheduler was produced and actually used to control an aircraft engine. The evaluation showed how the scheduler controlled the engine with an equivalent or improved stability. Also, the analysis provided better evidence for certification than by previous methods based on cyclic scheduling. Therefore, the approach was deemed to be successful and is awaiting an actual project to use it. The key finding of the case study is the high level of reuse obtained, the only changes made to the system were in the scheduler module, i.e. the rest of the software and hardware were unchanged.

As a consequence of the case study, the effects of the transition to fixed priority scheduling on the typical process life-cycle are investigated. Part of the case study included the use of the techniques by Rolls-Royce. Part of this use involved teaching engineers with no specialist knowledge of scheduling and timing analysis to apply the techniques. It was found that the engineers could understand and use the techniques on complex "real" examples within about an hour, which is a strong indication that the techniques were suitable. A number of issues, such as the need for more stringent requirements, were highlighted. A key issue is the ability to certify the approach. This was considered in enough detail to show the necessary evidence can be gathered. Part of the necessary evidence is: the approach is at least as safe as the existing approaches, and no additional hazards are introduced to the system as a whole. Other work by the author and colleagues [36, 101, 109] deals with the issue of certification of the approach.

Chapter 8 investigates how timing analysis may be performed for task sets where tasks have offsets. The approach is based on forming a composite task, for analysis purposes only, that has zero offsets and can replace the tasks with non-zero offsets. The approach is supplemented with a free variable argument that increases the value of task's offsets so that the analysis is less pessimistic. The free variable argument assigns regularly spaced slots to each task. The work has a number of aims, which are; to reuse the existing analysis, to provide an understandable technique, to have pseudo-polynominal computational complexity, and to achieve low pessimism. In this respect the approach can be considered novel. Part of this work investigated how the existing uniprocessor timing analysis can

be improved for the specific computational model being used. One of the principal benefits of the technique is the fact the existing schedulability analysis can be used with a limited amount of pre- and post-processing. There are a number of benefits of being able to reuse the analysis, including the fact existing tools and training can still be used as well as only a limited amount of extra certification evidence is needed.

Chapter 9 investigates how the scheduling and timing analysis derived in Chapters 4 - 8 can be used for distributed scheduling so that the maximum reuse of infrastructure and timing analysis may be attained. An approach is derived that makes use of offsets to control precedence in distributed transactions. Task attributes are derived that represent all the system's timing requirements, including the distributed ones. This allows verification to be performed an individual processor at a time. The timing analysis makes use of the offset analysis developed in Chapter 8. The approach developed is novel and achieves a number of goals. The goals are; robustness to change is improved, high levels of reuse can be attained, and pessimism is reduced compared to existing published work. Other work that the author has contributed to [109] has addressed the certification issues. Again, one of the principal benefits is the reuse of existing tools and training from the "standard" uniprocessor schedulability analysis with only a relatively small amount of pre- and post-processing.

## 10.1 Future Work

There are a number of areas in which further work could be performed. These are:

1. Investigate how the pessimism in the offset analysis can be improved by using other techniques (such as exact analysis or analysis that assumes a critical instant) than the composite analysis, where it is deemed preferable. The investigation would consider the conditions needed for the schedulability analysis where other analysis, would be better and when it is absolutely necessary to adopt exact analysis. An approach could be derived with lower pessimism that the composite approach with only a small increase in the average computational complexity.

2. In the context of distributed scheduling, the free variable uses "linear" time-slicing of the transaction's execution window when assigning the individual tasks' execution window. An investigation could be performed that looked at more flexible free variable arguments for cases when the existing mechanism results in an unschedulable solution. For each task in the transaction, a new execution window could be chosen based on the current laxity of the tasks in the transaction. Search mechanisms could be employed to find the best set of task attributes - best being schedulable with greatest robustness to change.

3. Investigation of an optimised free variable argument that selects the value for tasks' offsets in an iterative or intelligent manner rather than a simple linear equation. One way this could be achieved is to use a branch and bound search over the range of values of the offsets [minimum possible offset, maximum possible offset], and choose the "best" value. Best is judged by the likelihood a schedulable system is obtained, with secondary criteria of reduced computational complexity and robustness to change.

4. Examine how best to deal with distributed transactions that have arbitrary deadlines. Preliminary inspections has shown the current technique using offsets and composite analysis already supports distributed transactions with arbitrary deadlines. However, work is required to justify the statement and also to investigate the effectiveness attained.

## 10.2 Final Comment

In chapter 1, the following contention is made aimed at supporting the development and verification of schedulers for industrial safety critical hard real-time systems:

*The proposed simplified version of fixed priority scheduling will ease the problem of meeting timing requirements now, and for the immediate future for industrial safety critical embedded systems.*

To support this contention, four criteria were formed to guide the solutions so that successful technology transfer may be achieved. The criteria are: certification, sufficiency, understanding and reuse. Throughout all the work these criteria

have guided the solutions that were derived, and wherever possible these criteria are met. The following is a brief summary of how the approaches presented relate to the four criteria:

1. *Certification* - The timing analysis provides valuable evidence that can be used during the certification of the system and it can be argued that the changes made to the infrastructure do not lower the system's integrity. The approaches derived have been presented to the relevant certification authorities and their use has been approved on a future aircraft engine.

2. *Sufficiency* - The approaches allows tasks to be released efficiently (with reasonable overheads), effectively (with no jitter) and flexibly (with no restrictions of the task attributes, i.e. a small selection of possible iteration rates).

3. *Understanding* - The success of the work is reinforced by the fact the techniques have been transferred to industry. Rolls-Royce have themselves evaluated the techniques on an aircraft engine and have decided to use them on the next suitable project. During the evaluation, it was found that the engineers could understand and use the techniques on complex "real" examples within about an hour, which is a strong indication that the techniques were suitable. The distributed scheduling techniques is also being evaluated for a current Department of Trade and Industry funded industrial research project, and a Ministry Of Defence funded industrial research project.

4. *Reuse* - When the approaches were used by Rolls-Royce the only changes that were necessary to the system were to the software module for the scheduler (i.e. none of the applications or the actual hardware had to be altered). In addition, tools had to be produced for task attribute assignment and schedulability analysis. Their use of the approach demonstrates the high level of reuse that can be attained.

From the list above, it can be seen that the work has largely met its objectives.

# Bibliography


[1] "Military Standard 882C: System Safety Program Requirements," tech. rep., US Department of Defence, January 1993.

[2] A. Burns and J. A. McDermid, "Real-time safety-critical systems: analysis and synthesis," *Software Engineering Journal*, vol. 9, pp. 267–81, Nov. 1994. Softw. Eng. J. (UK).

[3] S. G. Hutchesson, "Multi-integrity level software on uni-processor systems," Master's thesis, Department of Computer Science, University of York, 1998.

[4] D. Turner, "Customer needs for future controls," in *The Design and Control of the Next Generation of Civil and Military Engines: Do Variable Cycle Engines Have a Role*, pp. 1.1–1.15, November 1994.

[5] H. Thompson, "Application of COTS technologies to aerospace gas turbine engine control," in *IEE Colloquium on COTS and Safety Critical Systems*, no. Digest Number: 97/013, January 1997.

[6] C. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives," *Real-Time Systems*, vol. 4, pp. 37–53, March 1992. Real-Time Syst. (Netherlands).

[7] Y. Yeh, "Dependability of the 777 primary flight control system," *5th IFIP Working Conference on Dependable Computing for Critical Applications*, 1995.

[8] R. Edwards and G. Parr, "Key issues in integrated modular avionics - IAWG viewpoint," in *ERA Avionics Conference*, pp. 5.1–5.12, 1994.

[9] *ARINC 653: Avionics Application Software Standard Interface (Draft 15).* Airlines Electronic Engineering Committee (AEEC), June 17th, 1996.

[10] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 40–61, 1973.

[11] T. Baker, "Task-based scheduling of real-time processes," *The Journal of Real-Time Systems*, vol. 3, no. 1, pp. 76–100, 1991.

[12] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini, "ARINC 659 scheduling problem," *IEEE Real-Time Systems Symposium*, 1994.

[13] L. Randazeese, "The overall incidence of transfer has been limited," *IEEE Transactions on Engineering Management*, vol. 43, pp. 393–401, November 1996.

[14] R. Bloeden, "Making university / industry collaborative research succeed," *Research Technology Management*, vol. 37, no. 2, pp. 44–48, 1994.

[15] R. Dorf, "Models for technology transfer from universities and research laboratories," *Technology Management*, no. 1, pp. 302–312, 1988.

[16] M. Torngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems*, vol. 14, pp. 219–250, May 1998.

[17] C. Locke and J. Goodenough, "Generic avionics software specification," Tech. Rep. CMU/SEI-90-TR-8, Software Engineering Institute, 1990.

[18] K. Wika and J. Knight, "On the enforcement of software safety policies," in *10th Annual IEEE Conference on Computer Assurance*, June 1995.

[19] D. N. Burghes and A. Graham, *Introduction to control theory, including optimal control.* Wiley, 1980.

[20] D. Thompson, *The Oxford Quick Reference Dictionary.* Oxford University Press, 1996.

[21] R. Luck, *Observability and delay compensation of integrated communication and control systems.* PhD thesis, Department of Mechanical Engineering, Pennslyvannia State University, USA, 1989.

[22] A. G. Shutler and H. Betteridge, "Definition, design and implementation of control laws for variable cycle gas turbine aircraft engines," in *The Design and Control of the Next Generation of Civil and Military Engines: Do Variable Cycle Engines Have a Role*, November 1994.

[23] A. Ray and Y. Helevi, "Integrated communication and control systems: Part 1 - analysis, and part 2 - design considerations," *ASME Journal of Dynamic Systems, Measurements and Control*, pp. 367–381, December 1988.

[24] B. Wittenmark, J. Nilsson, and M. Torngren, "Timing problems in real-time control systems: Problem formulation," in *Proceedings of the American Control Conference*, 1995.

[25] J. Rushby, "Kernel for safety?," in *Safe & Secure Computing Systems* (A. T., ed.), 1989.

[26] United Kingdom Ministry of Defence, *Defence Standard 00-55: Requirements for Safety-Related Software in Defence Equipment*, July 1996.

[27] J. A. McDermid and L. M. Barroca, "Formal methods: Use and relevance for the development of safety critical systems," *Computer Journal*, vol. 35(6), 1992.

[28] A. Burns, A. J. Wellings, C. Bailey, and E. Fyfe, "The olympus attitude and orbital control system, a case study in hard real-time system design and implementation," pp. 240–248, Springer Verlag, 1993.

[29] A. Burns and A. Wellings, "Safety kernels - specification and implementation," *The Design and Development of Safety Kernels*, vol. York Software Engineering for the Health and Safety Executive Nuclear Research Programme, 1994.

[30] C. Locke, *Best-Effort Decision Making for Real-Time Scheduling.* PhD thesis, Computer Science Department, Carnegie Mellon University, USA, 1986.

[31] A. Tannenbaum, *Computer networks.* Prentice Hall, 3 ed., 1996.

[32] United Kingdom Ministry of Defence, *Defence Standard 00-56 (Issue 2): Safety Management Requirements for Defence Systems*, December 1996.

[33] RTCA Inc., "Software considerations in airborne systems and equipment certification," DO-178B/ED-12B, December 1992.

[34] M. of Defence, "The procurement of safety critical software in defence equipment,interim defence standard, def-stan 00-55," tech. rep., April 1991.

[35] Y. Papadopoulos and J. A. McDermid, "The potential for a generic approach to the certification of safety critical systems in the transportation sector," in *Reliability Engineering and System Safety*, vol. 63, pp. 47–66, Elsevier, January 1999.

[36] I. J. Bate, A. Burns, J. A. McDermid, and A. J. Vickers, "Towards a fixed priority scheduler for an aircraft application," in *8th Euromicro Workshop on Real-Time Systems*, (L'Aqulia, Italy), pp. 34–39, June 1996.

[37] J. A. McDermid, *Software Engineer's Reference Book*. Butterworth Heinemann, 1991.

[38] I. Sommeville, *Software Engineering*. Addison Wesley, 5th ed., 1995.

[39] G. Booch and D. Bryan, *Software Engineering with Ada*. Benjamin Cummins, 3rd ed., 1994.

[40] B. Carre, "SPARK: the SPADE Ada kernel (edition 3.1)," tech. rep., Program Validation Limited, 1992.

[41] P. Puschner and C. Koza, "Calculating the maximum time of real-time programs," *Real-Time Systems*, vol. 1, no. 2, pp. 159–176, 1989.

[42] C. Park, "Predicting program execution times by analyzing static and dynamic program paths," *Real-Time Systems*, vol. 5, no. 1, pp. 31–62, 1993.

[43] C. Park and A. Shaw, "A source level tool for predicting deterministic execution times of programs," Tech. Rep. 89-09-02, Department of Computer Science and Engineering, University of Washington, USA, 1989.

[44] N. Leveson, *Safeware: System Safety and Computers*. Addison Wesley, 1995.

[45] M. Garey and D. Johnson, "Computers and intractability," 1979.

[46] A. Burns, N. Hayes, and M. Richardson, "Generating feasible cyclic schedules," *Control Engineering Practice*, vol. 3, no. 2, pp. 151–162, 1995.

[47] L. Sha, J. Lui, and J. Goodenough, "Real-time scheduling theory and ada," *Computer*, vol. 23, pp. 53–62, April 1990. Computer (USA).

[48] G. Carlow, "Architecture of the space shuttle primary avionics software system," *Communications ACM*, vol. 27, pp. 926–36, Sept. 1984. Commun. ACM (USA).

[49] O. Serling, "Scheduling of time critical processes," in *Proceedings AFIPS Spring Computing Conference*, 1972.

[50] J. Lehoczky, L. Sha, and Y. Ding, "The rate-monotonic scheduling algorithm: Exact characterization and average case behaviour," in *Proceedings IEEE Real-Time Systems Symposium*, pp. 166–171, 1989.

[51] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling: an historical perspective," *Real-Time Systems*, vol. 8, pp. 173–98, March-May 1995. Real-Time Syst. (Netherlands).

[52] D. Katcher, H. Arakawa, and J. Strosnider, "Engineering and analysis of fixed priority schedulers," *IEEE Trans. Software Engineering*, vol. 19, pp. 920–34, Sept. 1993. IEEE Trans. Softw. Eng. (USA).

[53] J. Y. T. Leung, "A note on preemptive scheduling of periodic real-time tasks," *Information processing Letters*, vol. 11, November 1980.

[54] N. C. Audsley, *Flexible Scheduling of Hard Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, December 1993.

[55] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceeeding of the Real-Time Systems Symposium*, pp. 201–209, December 1990.

[56] J. Harter, "Response times in level-structured systems," Tech. Rep. CU-CS-269-94, Department of Computer Science, University of Colorado, USA, 1984.

[57] P. Harter, "Response times in level-structured systems," *ACM Trans. Computer Systems*, vol. 5 A02, pp. 232–248, Aug. 1987. ACM Trans. Comput. Syst. (USA).

[58] M. Joseph, "On a problem in real-time computing," *Information Processing Letters*, vol. 20, no. 4, pp. 173–177, 1985.

[59] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Computer Journal*, vol. 29 A02, pp. 390–5, Oct. 1986. Comput. J. (UK).

[60] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline monotonic approach," in *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, pp. 127–132, 1991.

[61] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation (Netherland)*, vol. 2, no. 4, pp. 237–250, 1982.

[62] K. Tindell, *Holistic Scheduling Analysis for Distributed Hard Real-Time Systems*. No. YCS 197, Department of Computer Science, University of York, 1993.

[63] R. Gerber, S. Hong, and M. Sabsena, "Guaranteeing end-to-end timing constraints by calibrating intermediate processes," *IEEE Real-Time Systems Symposium*, 1994.

[64] R. Yerraballi, *Scalability in Real-Time Systems*. PhD thesis, Computer Science Department, Old Dominion University, August 1996.

[65] B. Lampson and D. Redell, "Experience with processes and monitors in mesa," *Communications ACM*, vol. 23, pp. 105–17, Feb. 1980. Commun. ACM (USA).

[66] R. Rajkumar, L. Sha, and J. Lehoczky, "Real-time synchronisation protocols for multiprocessors," *IEEE Real-Time Systems Symposium*, pp. 259–269, 1988.

[67] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Trans. Computers*, vol. 39, pp. 1175–85, Sept. 1990. IEEE Trans. Comput. (USA).

[68] R. Rajkumar, L. Sha, J. Lehoczky, and K. Ramamritham, "An optimal priority inheritance protocol for real-time synchronisation," Tech. Rep. Coins Technical Report 88-98, 1988.

[69] J. A. Clark and K. Tindell, "Holistic schedulability analysis for distributed hard real time systems," *Microprocessing & Microprogramming*, vol. 50, pp. 117–134, April 1994.

[70] J. Gutierrez, J. Garcia, and M. Harbour, "On the schedulability analysis for distributed real-time systems," in *9th Euromicro Workshop on Real-Time Systems*, pp. 136–143, 1997.

[71] A. Burns, K. Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers," *IEEE Transactions on Software Engineering*, vol. 21, pp. 475–480, May 1995.

[72] A. Burns, A. J. Wellings, and E. Fyfe, *The Olympus Attitude and Orbital Control System, A Case Study in Hard Real-time System Design and Implementation YCS 190.* Dept of Computer Science, University of York C. Bailey British Aerospace Space Systems Ltd, January 1993.

[73] J. Sun and J. Liu, "Synchronization protocols in distributed real-time systems," in *In The 16th International Conference on Distributed Computing Systems*, May 1996.

[74] R. Bettati and J.-S. Liu, "End-to-end scheduling to meet deadlines in distributed systems," in *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pp. 452–459, June 1992.

[75] M. Di Natale and J. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 216–227, December 1994.

[76] K. G. Shin and J. Jonsson, "Robust adaptive metrics for deadline assignment in distributed hard real-time systems," *Submitted to IEEE Transactions on Computers*, 1998.

[77] J. Garcia and M. Harbour, "Optimized priority assignment for tasks and messages in distributed real-time systems," *IEEE Parallel and Distributed Systems*, pp. 124–131, 1995.

[78] E. H. L. Aarts and J. H. M. Korst, *Simulated Annealing and Boltzmann Machines*. Chichester, U.K.: John Wiley and Sons, 1989 techniques.

[79] K. Whitely, "A genetic algorithm tutorial," Tech. Rep. CS-93-103, 1993.

[80] H. Kopetz, *Real-Time Systems. Design Principles for Distributed Embedded Applications*. Kluwer Academic Publications, 1997.

[81] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schutz, "The design of real-time systems: from specification to implementation and verification," *Software Engineering Journal*, vol. 6, pp. 72–82, May 1991. Softw. Eng. J. (UK).

[82] H. Kopetz, G. Grunsteidl, and J. Reisinger, "Fault tolerant membership service in a synchronous distributed real-time systems," Tech. Rep. 4/89, Institut fur Technische Informatik, Technische Univeristat Wien, February 1989.

[83] R. A. Edwards, "ASAAC phase I harmonized concept summary," in *Proceedings ERA Avionics Conference and Exhibition*, (London, UK), 1994.

[84] J. Gosling, B. Joy, and G. Steel, *The Java Language Specification*. Addison-Wesley, 1996.

[85] M. Fletcher, A. Wake, and J. Bradley, "Integrated modular avionics and certification - an ima design team's view," in *Seminar on Certification of Ground/Air Systems*, The Institute of Electrical Engineers, February 1998.

[86] A. Grigg and N. C. Audsley, "Towards the timing analysis of integrated modular avionics systems," *Proceedings ERA Avionics Conference and Exhibition*, 1997.

[87] K. G. Shin and Y.-C. Chang, "A reservation based algorithm for scheduling both periodic and aperiodic real-time tasks," *IEEE Transactions on Computers*, vol. 44, pp. 1405–1491, December 1995.

[88] J. Stankovic, "Real-time computer systems: The next generation," Tech. Rep. Coins Technical Report Number 88-06, 1988.

[89] N. C. Audsley, K. W. Tindell, and A. Burns, "The end of the road for static cyclic scheduling," *Proceedings of 5th Euromicro Workshop on Real-Time Systems*, pp. 36–41, 1993.

[90] US Department of Defence, *Reference Manual for the Ada Programming Language*, 1983. ANSI/MUL-STD 1815.

[91] *Ada 95 Language Reference Manual.* Intermetrics Inc., 1995. ISO/IEC 8652:1995.

[92] J. Barnes, *High Integrity Ada: The SPARK Approach.* Addison-Wesley, 1997.

[93] ISO WG9, *Guidance for the Use of the Ada Programming Language in High Integrity Systems*, 3.5 ed., September 1998.

[94] J. A. McDermid, "COTS: The expensive solution?," in *IEE Colloquium on COTS and Safety Critical Systems*, no. Digest Number: 97/013, January 1997.

[95] A. Burns and A. J. Wellings, *Real-Time Systems and Programming Languages.* Addison Wesley, 2nd ed., 1997.

[96] S. Hutchesson and N. Hayes, "Technology transfer and certification issues in safety critical real-time systems," in *Digest of the IEE Colloquium on Real-Time Systems*, no. 98/306, April 1998.

[97] R. Campbell and B. Randell, "Error recovery in asynchronous systems," *IEEE Trans. Software Engineering*, vol. vol.SE-12, no.8 A02, pp. 811–26, Aug. 1986. IEEE Trans. Softw. Eng. (USA).

[98] A. Bertossi and L. Mancini, "Scheduling algorithms for fault-tolerance in hard-real-time systems," *Real-Time Systems*, vol. 7, pp. 229–45, Nov. 1994. Real-Time Syst. (Netherlands).

[99] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Trans.*

*Software Engineering*, vol. 21, pp. 579–92, July 1995. IEEE Trans. Softw. Eng. (USA).

[100] J. Gutierrez, J. Garcia, and M. Harbour, "Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems," in *10th Euromicro Workshop on Real-Time Systems*, pp. 35–44, 1998.

[101] N. C. Audsley, I. J. Bate, and A. Burns, "Putting fixed priority scheduling into engineering practice for safety critical applications," in *Proceedings of the Real-Time Technology and Applications Symposium*, pp. 2–10, IEEE Technical Committee on Real-Time Systems, 1996.

[102] I. Bate and A. Burns, "Flexible scheduling for engine controllers," Tech. Rep. UK Patent Application Number 9710522.5 and US Patent Application Number 5,606,695, The Patent Office, May 1997.

[103] N. Audsley, I. Bate, and A. Burns, "Flexible scheduling theory for advanced engine controllers," in *IEE Colloquia on Hybrid Control for Real-Time Systems*, Institute of Electrical Engineers, December 1996.

[104] S. Wilson, J. A. McDermid, P. Fenelon, and P. Kirkham, "No more spineless safety cases: A structured method and comprehensive tool support.," in *2nd International Conference on Control and Instrumentation in Nuclear Installations*, Institution of Nuclear Engineers, April 1995.

[105] K. Tindell, *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, Department of Computer Science, University of York, 1994.

[106] Y. Oh and S. Son, "A processor-efficient scheme for supporting fault-tolerance in rate-monotonic scheduling," Tech. Rep. CS-95-02, Department of Computer Science, University of Virginia, 1995.

[107] G. Fohler and C. Koza, "Heuristic scheduling for distributed real-time systems," Tech. Rep. Research Report No. 6, Institut fur technische Informatik, Technische Universtate Wien, Austria, 1989.

[108] J. Knight, A. Cass, A. Fernandez, and K. Wika, "Fixed priority scheduling of periodic tasks on multiprocessor systems," Tech. Rep. CS-94-08, Department of Computer Science, University of Virginia, 1994.

[109] I. J. Bate, A. Burns, T. P. Kelly, and J. A. McDermid, "Building a preliminary safety case: An example from aerospace," in *Proceedings of the 1997 Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, October 1997.