Synchronization Protocols
for a Compositional Real-Time Scheduling Framework

Moris Behnam

2010
SYNCHRONIZATION PROTOCOLS
FOR A COMPOSITIONAL REAL-TIME SCHEDULING FRAMEWORK

Moris Behnam

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras måndagen 22 november, 2010, 10.00 i Delta, Mälardalens högskola, Västerås.

Fakultetsopponent: prof. Giuseppe Lipari, Scuola Superiore Sant’Anna

Akademin för innovation, design och teknik
Abstract

In this thesis we propose techniques to simplify the integration of subsystems while minimizing the overall amount of CPU resources needed to guarantee the schedulability of real-time tasks. In addition, we provide solutions to the problem of allowing for the use of logical resources requiring mutual exclusion.

The contribution of the thesis is presented in three parts. In the first part, we propose a synchronization protocol, called SIRAP, to facilitate sharing of logical resources in a hierarchical scheduling framework. In addition, we extend an existing synchronization protocol, called HSRP, such that each subsystem can be developed independently. The performance of the proposed protocols is evaluated by extensive simulations. In the second part, we present an efficient schedulability analysis that exploits the lower scheduling overhead introduced by each of the proposed protocols. Finally, in the third part, we propose new methods and algorithms that find the optimal system parameters (e.g., optimal resource ceiling), that minimize the amount of CPU resources required to ensure schedulability, when using the proposed synchronization protocols in a hierarchical scheduling framework.

The motivation of this work comes from an emerging industrial trend in embedded software development to integrate multiple applications (subsystems) on a small number of processors. The purpose of this integration is to reduce the hardware related costs as well as the communication complexity between processors. In this setting a large number of industrial applications face the problem of preserving their real-time properties after their integration onto a single processor. An additional motivation is that temporal isolation between the applications during runtime may be required to prevent failure propagation between different applications.

Specifically, we propose a hierarchical scheduling framework that allows for a simplified integration of subsystems. The framework preserves the essential temporal characteristics of the subsystems, both when running in isolation as well as when they are integrated with other subsystems. In this thesis, we assume a model where a system consists of a number of subsystems. The subsystems can interact with each other using shared logical resources. The framework ensures that the individual subsystem respects its allocated share of the processor. The difficulty lies in allowing two or more subsystems to share logical resources, which introduces an additional complexity in the schedulability analysis and also increases the system load.
Abstract

In this thesis we propose techniques to simplify the integration of subsystems while minimizing the overall amount of CPU resources needed to guarantee the schedulability of real-time tasks. In addition, we provide solutions to the problem of allowing for the use of logical resources requiring mutual exclusion.

The contribution of the thesis is presented in three parts. In the first part, we propose a synchronization protocol, called SIRAP, to facilitate sharing of logical resources in a hierarchical scheduling framework. In addition, we extend an existing synchronization protocol, called HSRP, such that each subsystem can be developed independently. The performance of the proposed protocols is evaluated by extensive simulations. In the second part, we present an efficient schedulability analysis that exploits the lower scheduling overhead introduced by each of the proposed protocols. Finally, in the third part, we propose new methods and algorithms that find the optimal system parameters (e.g., optimal resource ceiling), that minimize the amount of CPU resources required to ensure schedulability, when using the proposed synchronization protocols in a hierarchical scheduling framework.

The motivation of this work comes from an emerging industrial trend in embedded software development to integrate multiple applications (subsystems) on a small number of processors. The purpose of this integration is to reduce the hardware related costs as well as the communication complexity between processors. In this setting a large number of industrial applications face the problem of preserving their real-time properties after their integration onto a single processor. An additional motivation is that temporal isolation between the applications during runtime may be required to prevent failure propagation between different applications.

Specifically, we propose a hierarchical scheduling framework that allows for a simplified integration of subsystems. The framework preserves the essential temporal characteristics of the subsystems, both when running in isolation
as well as when they are integrated with other subsystems. In this thesis, we assume a model where a system consists of a number of subsystems. The subsystems can interact with each other using shared logical resources. The framework ensures that the individual subsystem respects its allocated share of the processor. The difficulty lies in allowing two or more subsystems to share logical resources, which introduces an additional complexity in the schedulability analysis and also increases the system load.
as well as when they are integrated with other subsystems. In this thesis, we assume a model where a system consists of a number of subsystems. The subsystems can interact with each other using shared logical resources. The framework ensures that the individual subsystem respects its allocated share of the processor. The difficulty lies in allowing two or more subsystems to share logical resources, which introduces an additional complexity in the schedulability analysis and also increases the system load.

To the memory of my mother
Acknowledgment

This thesis would not have been possible without the help of my supervisors Thomas Nolte and Mikael Sjödin and the collaboration with Reinder Bril and Insik Shin. Thomas, thank you very much for the supporting, encouraging, helping and always finding time to guide me. I would like to thank Mikael Sjödin for his advices and invaluable input to my research. I would like to express my special gratitude to Reinder J. Bril for the successful collaboration and for his constructive comments and discussions. A special thank you goes to Insik for all the intensive discussions and fruitful cooperation. I would like to say how much I have appreciated working with Thomas, Reinder, Insik and Mikael, and I have learned a lot from them.

I would like to express my gratitude to all my co-authors for your collaboration, guidance, discussions and nice results, I really enjoyed working with you. Thanks to Farhang Nemati, Mikael Åsberg, Rui Santos, Martijn van den Heuvel.

I would like to thank the PROGRESSers; Hans Hansson for his great leading of the PROGRESS/MRTC center, and Ivica Crnkovic, Christoffer Norström, Sasikumar Punnekkat, Paul Pettersson, Jan Gustafsson, Andreas Ermedahl, Kristina Lundqvist, Cristina Seceleanu, and Jukka Mäki-Turja.

I would like to acknowledge Radu Dobrin and Antonio Cicchetti for reviewing my thesis and providing valuable feedback.

I would also like to thank my colleagues at the department for the nice time that we had in the department and during conference trips, project trips and PhD schools. I wish to give many thanks to Huseyin Aysan, Andreas Hajertström, Séverine Sentilles, Aneta Vulgarakis, Marcelo Santos, Stefan Bygde, Yue Lu, Jagadish Suryadevara, Aida Causevic, Rafia Inam, Kathrin Dannmann, Ana Petricic, Sara Dersten, Adnan Causevic, Nikola Petrovic, Holger Kienle,
Acknowledgment

This thesis would not have been possible without the help of my supervisors Thomas Nolte and Mikael Sjödin and the collaboration with Reinder Bril and Insik Shin. Thomas, thank you very much for the supporting, encouraging, helping and always finding time to guide me. I would like to thank Mikael Sjödin for his advice and invaluable input to my research. I would like to express my special gratitude to Reinder J. Bril for the successful collaboration and for his constructive comments and discussions. A special thanks goes to Insik for all the intensive discussions and fruitful cooperation. I would like to say how much I have appreciated working with Thomas, Reinder, Insik and Mikael, and I have learned a lot from them.

I would like to express my gratitude to all my co-authors for your collaboration, guidance, discussions and nice results, I really enjoyed working with you. Thanks to Farhang Nemati, Mikael Åsberg, Rui Santos, Martijn van den Heuvel.

I would like to thank the PROGRESSers; Hans Hansson for his great leading of the PROGRESS/MRTC center, and Ivica Crnkovic, Christer Norström, Sasikumar Punnekkat, Paul Pettersson, Jan Gustafsson, Andreas Ermedahl, Kristina Lundqvist, Cristina Seceleanu, and Jukka Mäki-Turja.

I would like to acknowledge Radu Dobrin and Antonio Cicchetti for reviewing my thesis and providing valuable feedback.

I would also like to thank my colleagues at the department for the nice time that we had in the department and during conference trips, project trips and PhD schools. I wish to give many thanks to Hüseyin Aysan, Andreas Häjertström, Séverine Sentilles, Aneta Vulgarakis, Marcelo Santos, Stefan Bygde, Yue Lu, Jagadish Suryadevara, Aida Causevic, Rafia Inam, Kathrin Dannmann, Ana Petricic, Sara Dersten, Adnan Causevic, Nikola Petrovic, Holger Kienle,
Federico Ciccozzi, Saad Mubeen, Mehrdad Saadatmand, Johan Kraft, Juraj Feljan, Luka Lednicki, Leo Hatvani, Josip Maras, Tiberiu Seceleanu, Etienne Borde, Thomas Leveque, Andreas SG Gustavsson, Batu Akan, Fredrik Ekstrand, Jörgen Lidholm, Giacomo Spampinato, Markus Bohlin, Jan Carlson, Stefan Cedergren, Barbara Gallina, Andreas Johnsen, Eun-Young Kang, Dag Nyström, Peter Wallin, Johan Fredriksson and Daniel Sundmark.

Special thanks goes to the administrative staff, in particular Gunnar Widforss, Malin Rosqvist, Åsa Lundkvist, Else-Maj Silén, Susanne Fronnå and Carola Ryttersson.

Many thanks go to Damir Isovic for informing me about the PhD position and for the very nice recommendation letter that I received from him when I applied for the position.

I would like to thank Nathan Fisher for hosting me at Wayne State University. I would also like to thank Pradeep Hettiarachchi for helping me during my visit to Wayne State University, it was very kind of you.

Finally, my deepest gratitude goes to my wife Rasha and my kids Dany and Hanna for all their support and love.

This work has been supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS, and the Swedish Research Council.

Moris Behnam
Västerås, November, 2010
List of Publications

Publications included in this thesis


**Publications relevant to the thesis but not included**

A) Journal & Conferences:


• Moris Behnam, Insik Shin, Thomas Nolte, Mikael Nolin, *An Overrun Method to Support Composition of Semi-Independent Real-Time Components*, In Proceedings of the 32\(^{nd}\) Annual IEEE International Com-

B) Workshops:


Notes for the Reader

This thesis contains two parts. The first part is an introductory part included in chapters 1-5. The second part is a collection of seven papers (A-G) in chapters 6-12. The seven papers are structured in 3 sections as follows:

- Hierarchical scheduling and synchronization (papers A-C).
- Schedulability analysis (papers D-E).
- Algorithms for efficient CPU resource usage (papers F-G).

Note that throughout the seven papers, there are some differences in notations, indexes and terminologies. For instance, in some papers we use the term resource holding time and in other papers we use resource locking time for the same thing. In addition, in some papers we assume that tasks are sorted according to their priorities, in the order of increasing priority, and in other papers we assume that they are sorted in the order of decreasing priority. Therefore it is important to read and follow the corresponding system model of each paper, respectively. Finally, it is recommended to read all included papers before reading chapter 4 (Summary, Conclusions and Future Work) for a better understanding of this chapter.
Swedish Summary


En tydlig trend inom många programvaruintensiva industriella tillämpningsområden, till exempel bilindustrin och flygindustrierna, är att integrera flera delsystem på ett mindre antal processorer. Syftet med denna integrering är dels att minska olika typer av kostnader, samt att minska komplexiteten framförallt med avseende förenkling av kommunikation mellan delsystem som inte längre behöver ske över fysiska nätverk.


Vi föreslår ett hierarkiskt schemaläggningsramverk som möjliggör en förenklad integrationsprocess av delsystem. Detta ramverk bevarar viktiga tidsmässiga egenskaper hos delsystemen, både när dessakör isolerat och när de är integrerade tillsammans med andra delsystem.

I denna avhandling utgår vi ifrån en modell där ett system består av ett antal delsystem. Delsystemen kan interagera med varandra med hjälp av delade logiska resurser. Ramverket ser till att de enskilda delsystemen respekterar sin tilldelade andel av processorn. Svårigheten ligger i att tillåta två eller flera delsystem att tillsammans använda processorn.
Målet med avhandlingen är att förenkla integration av delsystem och samtidigt minimera processorkraften som krävs för att delsystemen ska hinna med att utföra alla uppgifter på ett tillfredsställande sätt. I den här avhandlingen fokuserar vi på problemet med att tillåta användandet av logiska resurser tillsammans med hierarkisk schemaläggning och vi föreslår ett nytt synkroniseringsprotokoll för detta. Vi föreslår även nya algoritmer och analyser som på ett resurseffektivt sätt kan minimera processorbelastning vid användandet av synkroniseringsprotokoll för hierarkiska schemaläggare.

En tydlig trend inom många programvaruintensiva industriella tillämpningsområden, till exempel bilindustrin och flygindustrin, är att integrera flera delsystem på ett mindre antal processorer. Syftet med denna integrering är dels att minska olika typer av kostnader, samt att minska komplexiteten framförallt med avseende förenkling av kommunikation mellan delsystem som efter integrering inte längre behöver ske över fysiska nätverk.


Vi föreslår ett hierarkiskt schemaläggningsramverk som möjliggör en förenklad integrationsprocess av delsystem. Detta ramverk bevarar väsentliga tidsmässiga egenskaper hos delsystemen, både när dessa kör isolerat och när de är integrerade tillsammans med andra delsystem.

I denna avhandling utgår vi ifrån en modell där ett system består av ett antal delsystem. Delsystemen kan interagera med varandra med hjälp av delade logiska resurser. Ramverket ser till att de enskilda delsystemen respekterar sin tilldelade andel av processorn. Svårigheten ligger i att tillåta två eller flera
delsystem att dela logiska resurser, vilket introducerar en extra komplexitet i schemaläggningsanalysen och dessutom ökar processorns belastning.
## Contents

### I Thesis

#### 1 Introduction

1.1 Contributions ........................................ 5
   1.1.1 Hierarchical scheduling and synchronization .... 5
   1.1.2 Schedulability analysis ................................ 6
   1.1.3 Algorithms for efficient CPU resource usage .... 7
1.2 Outline of thesis ...................................... 7

#### 2 Background and System Model

2.1 Real-time systems ...................................... 9
   2.1.1 Scheduling algorithms ................................ 10
   2.1.2 Logical resource sharing ............................ 11
2.2 System model .......................................... 12

#### 3 A Real-Time Hierarchical Scheduling Framework with Logical Resource Sharing

3.1 HSF schedulability analysis ............................. 16
   3.1.1 Virtual processor model ............................. 16
   3.1.2 Local schedulability analysis ....................... 17
   3.1.3 System composability ............................... 18
   3.1.4 Subsystem interface evaluation ................. 18
3.2 Global resource sharing ............................... 19
   3.2.1 Problem formulation ............................... 20
3.3 Supporting global resource sharing .................... 21
   3.3.1 SIRAP ........................................... 22
   3.3.2 HSRP ........................................... 24
   3.3.3 The BROE server ................................... 24
## 3.3.4 BWI

3.4 Isolation between subsystems

3.5 Comparing SIRAP, HSRP and BROE

  3.5.1 Theoretical comparison
  3.5.2 Implementation complexity and overhead

## 4 Summary, Conclusions and Future Work

  4.1 Conclusions
  4.1.1 Discussion
  4.2 Future work

## 5 Overview of the Papers

  5.1 Paper A
  5.2 Paper B
  5.3 Paper C
  5.4 Paper D
  5.5 Paper E
  5.6 Paper F
  5.7 Paper G

Bibliography

## II Included Papers


  6.1 Introduction
  6.2 Related work
  6.3 System model
    6.3.1 Hierarchical scheduling framework
    6.3.2 Shared resources
    6.3.3 Virtual processor model
    6.3.4 Subsystem model
  6.4 SIRAP protocol
    6.4.1 Terminology
    6.4.2 SIRAP protocol description
  6.5 Schedulability analysis
    6.5.1 Local schedulability analysis
### Paper C:
**Overrun and Skipping in Hierarchically Scheduled Real-Time Systems**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>117</td>
</tr>
<tr>
<td>8.2</td>
<td>Related work</td>
<td>118</td>
</tr>
<tr>
<td>8.3</td>
<td>System model and background</td>
<td>120</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Shared resources</td>
<td>122</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Schedulability analysis</td>
<td>123</td>
</tr>
<tr>
<td>8.4</td>
<td>Comparing overrun and skipping</td>
<td>126</td>
</tr>
<tr>
<td>8.4.1</td>
<td>System load</td>
<td>126</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Experiment definition</td>
<td>127</td>
</tr>
<tr>
<td>8.4.3</td>
<td>Simulation results</td>
<td>128</td>
</tr>
<tr>
<td>8.5</td>
<td>Summary</td>
<td>131</td>
</tr>
</tbody>
</table>

Bibliography | 133 |

### Paper D:
**Bounding the Number of Self-Blocking Occurrences of SIRAP**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>139</td>
</tr>
<tr>
<td>9.2</td>
<td>Related work</td>
<td>140</td>
</tr>
<tr>
<td>9.3</td>
<td>System model and background</td>
<td>141</td>
</tr>
<tr>
<td>9.4</td>
<td>SIRAP</td>
<td>143</td>
</tr>
<tr>
<td>9.5</td>
<td>Motivating example</td>
<td>147</td>
</tr>
<tr>
<td>9.6</td>
<td>Improved SIRAP analysis</td>
<td>148</td>
</tr>
<tr>
<td>9.6.1</td>
<td>Problem formulation</td>
<td>149</td>
</tr>
<tr>
<td>9.6.2</td>
<td>Self-blocking set</td>
<td>150</td>
</tr>
<tr>
<td>9.6.3</td>
<td>Analysis based on changing $rbf$</td>
<td>150</td>
</tr>
<tr>
<td>9.6.4</td>
<td>Analysis based on changing $sbf$</td>
<td>155</td>
</tr>
<tr>
<td>9.7</td>
<td>Evaluation</td>
<td>159</td>
</tr>
<tr>
<td>9.7.1</td>
<td>Simulation settings</td>
<td>160</td>
</tr>
<tr>
<td>9.7.2</td>
<td>Simulation results</td>
<td>160</td>
</tr>
<tr>
<td>9.8</td>
<td>Summary</td>
<td>165</td>
</tr>
</tbody>
</table>

Bibliography | 167 |

### Paper E:
**Improved Schedulability Analysis of Synchronization Protocols Based on Overrun Without Payback for Hierarchical Scheduling Frameworks**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>171</td>
</tr>
<tr>
<td>10.1.1</td>
<td>Background</td>
<td>171</td>
</tr>
</tbody>
</table>
10.1.2 Contributions .............................................. 172
10.1.3 Overview ................................................. 172
10.2 Related work .............................................. 172
10.3 Real-time scheduling model ................................. 173
  10.3.1 System model ........................................... 173
  10.3.2 Subsystem model ........................................ 174
  10.3.3 Task model .............................................. 174
  10.3.4 Resource model ......................................... 174
  10.3.5 Synchronization protocol .............................. 175
10.4 Recap of existing schedulability analysis ................. 176
  10.4.1 Global analysis .......................................... 176
  10.4.2 Local analysis .......................................... 178
10.5 Improved global analysis .................................. 178
  10.5.1 Illustrating the improvement ............................ 178
  10.5.2 Improving the global analysis ......................... 179
  10.5.3 Concluding remarks .................................... 182
10.6 Improved local analysis ................................... 184
10.7 Evaluation .................................................. 184
  10.7.1 System load ............................................. 185
  10.7.2 Simulation setting ...................................... 187
  10.7.3 Simulation results ...................................... 188
10.8 Conclusion .................................................. 193
Bibliography ..................................................... 195

11 Paper F: Refining SIRAP with a Dedicated Resource Ceiling for Self-Blocking 199
11.1 Introduction ................................................ 201
11.2 Related work .............................................. 202
11.3 System model and background ............................. 203
11.4 SIRAP ....................................................... 206
11.5 Improved SIRAP analysis ................................. 209
11.6 Improved SIRAP protocol .................................. 209
  11.6.1 Subsystem ceiling for self-blocking .................. 210
  11.6.2 Subsystem ceiling upon self-blocking ............... 212
11.7 Selection algorithm ....................................... 214
11.8 Algorithm evaluation ..................................... 218
  11.8.1 Simulation settings .................................... 218
  11.8.2 Simulation results ..................................... 219
11.9 Summary ..................................................... 222
Bibliography ............................................................. 223

12 Paper G: 
Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources 227

12.1 Introduction ....................................................... 229
12.2 Related work .................................................... 230
12.3 System model and background ................................. 231
  12.3.1 Virtual processor models ................................ 231
  12.3.2 System model .............................................. 232
  12.3.3 Stack resource policy (SRP) ............................. 233
12.4 Resource sharing in the HSF ................................. 234
  12.4.1 Overrun mechanism ....................................... 234
  12.4.2 Schedulability analysis .................................. 235
12.5 Problem formulation and solution outline ..................... 236
12.6 Interface candidate generation .............................. 238
  12.6.1 ICG algorithm ............................................. 242
12.7 Interface selection ............................................. 244
  12.7.1 Description of the ICS algorithm ....................... 244
  12.7.2 Correctness of the ICS algorithm ....................... 247
12.8 Conclusion ...................................................... 253
Bibliography ............................................................. 255
I

Thesis
Chapter 1

Introduction

In this thesis we address the challenges of allowing sharing of logical resources between tasks that are scheduled by a Hierarchical Scheduling Framework (HSF). Given this HSF, our aim is to provide an efficient compositional integration framework, in terms of CPU resources required to preserve temporal behavior for independently developed applications (subsystems) executing on a single processor.

Motivation

The complexity of embedded systems is increasing exponentially due to requirements on advanced functionality. For example in the automotive domain, functionality that was realized by mechanical subsystems is often partially or completely replaced by embedded systems (for example engine control, anti-lock braking, etc.). Also new and advanced functionalities are required to be added (for example collision avoidance system, car to car communication, steer by wire, brake by wire, etc.).

To deal with the high complexity of embedded systems, systems are today developed as a set of independent subsystems often by different suppliers. In the final development stages, these subsystems are integrated to produce the final product. Traditionally, in many software intensive industrial application domains, such as automotive and avionics, each subsystem is assigned to one or more dedicated Electronic Control Units (ECUs). In order to provide isolation between subsystems during runtime, different subsystems are not allowed to be executed on the same ECU. However, with the increase of functionality, this approach significantly increases the complexity of the embedded systems in terms of requiring a high number of ECUs, with complex communication so...
Chapter 1

Introduction

In this thesis we address the challenges of allowing sharing of logical resources between tasks that are scheduled by a Hierarchical Scheduling Framework (HSF). Given this HSF, our aim is to provide an efficient compositional integration framework, in terms of CPU resources required to preserve temporal behavior for independently developed applications (subsystems) executing on a single processor.

Motivation  The complexity of embedded systems is increasing exponentially due to requirements on advanced functionality. For example in the automotive domain, functionality that was realized by mechanical subsystems is often partially or completely replaced by embedded systems (for example engine control, anti-lock braking, etc.). Also new and advanced functionalities are required to be added (for example collision avoidance system, car to car communication, steer by wire, brake by wire, etc.).

To deal with the high complexity of embedded systems, systems are today developed as a set of independent subsystems often by different suppliers. In the final development stages, these subsystems are integrated to produce the final product. Traditionally, in many software intensive industrial application domains, such as automotive and avionics, each subsystem is assigned to one or more dedicated Electronic Control Units (ECUs). In order to provide isolation between subsystems during runtime, different subsystems are not allowed to be executed on the same ECU. However, with the increase of functionality, this approach significantly increases the complexity of the embedded systems in terms of requiring a high number of ECUs, with complex communication so-
olutions in between ECUs. To reduce complexity and cost of these systems, one current trend is to integrate more software subsystems into a lower number of processors [1]. One example can be integrating both the engine control and the gearbox subsystems in one ECU. However, many subsystems have real-time requirements which raise the problem of guaranteeing the timing behavior of these subsystems also after integrating them in a single processor. In addition, temporal isolation between the subsystems during runtime is required to prevent one application from causing a failure of another subsystem.

The hierarchical scheduling framework has been introduced to enable compositional schedulability analysis of systems with real-time constraints to simplify schedulability analysis of complex systems [2]. It offers many additional interesting features that can solve the problem of guaranteeing temporal requirements during the integration of independently developed applications in a single processor. The HSF provides means for decomposing a complex software system into well-defined parts (subsystems). Each subsystem is associated with an abstract notion of its total CPU resource requirements. This abstract notion, manifested by the subsystem timing interface, is used during subsystem design time for various kinds of analysis, and during runtime to guarantee correct allocation of CPU resources to the system. In this thesis we refer to this kind of interface-based hierarchical scheduling as the Hierarchical Scheduling Framework (HSF). The main feature of the HSF is that it provides CPU partitioning between different subsystems. Thus, subsystems can be isolated from each other for, e.g., fault containment, compositional verification, validation and certification, unit testing, independent development etc. Finally, since subsystems can be developed independently, the HSF facilitates reusability of subsystems in systems that have real-time constraints.

Integrating different subsystems in a single processor implies that these subsystems will not only share the CPU resources, but they may also be in direct competition for other types of resources (such as flash memory, a memory map of a peripheral device, data structures etc.). Many of these resources may be accessed in a non-preemptable manner (using mutual exclusion). Resources that are shared by tasks (in a non-preemptable manner) from different subsystems are called global shared resources, and synchronization protocols should be used to synchronize the access to these shared resources. However, traditional synchronization protocols such as the Priority Inheritance Protocol (PIP) [3], the Priority Ceiling Protocol (PCP) [4], and the Stack Resource Policy (SRP) [5], give rise to a problem of excessive blocking of subsystems due to budget depletion during global shared resource access (more details will be explained in Chapter 3). More appropriate protocols are needed for hierarchical
scheduling frameworks.
In this thesis, our overall goal is to propose a HSF and corresponding synchronization protocols that together are able to fulfill the following requirements:

- The HSF should support sharing of logical resources between subsystems while preserving temporal predictability.
- The HSF should support independent development of subsystems. This requirement enables parallel development of subsystems, as different suppliers can develop different subsystems without revealing the internal details of each subsystem. In addition, this requirement facilitates reuse of software legacy systems/subsystems; systems that have been developed for a long time possibly not complying with any particular system model.
- The HSF should use CPU-resources efficiently. This requirement can be achieved by minimizing system load, the collective CPU needed to guarantee the schedulability of the entire framework. This requirement is a very important since fulfilling the first two requirements, increases the systems load (this will be explained in more details in Chapter 3).

1.1 Contributions

The contributions presented in this thesis can be divided into three parts:

1.1.1 Hierarchical scheduling and synchronization
As mentioned above, traditional synchronization protocols such as PIP, PCP and SRP can not handle the problem of resource sharing in hierarchical scheduling frameworks. Hence, more advanced protocols are needed for this kind of systems.

- In paper A we present Subsystem Integration and Resource Allocation Policy (SIRAP); a synchronization protocol for hierarchical scheduling. In addition, we present a simple schedulability analysis that bounds the timing behavior of SIRAP.
- In paper B we develop a schedulability analysis of an existing synchronization protocol HSRP, such that it allows for independent analysis of
individual subsystems. To distinguish between the original analysis of HSRP and the proposed analysis, we use the term *Overrun* to refer to the proposed analysis.

- Finally, in paper C we present a comparative evaluation of the *Overrun* and SIRAP by means of simulation. We apply the protocols on the HSF and we use the same system settings allowing for a fair comparison. The simulation results indicate when one protocol is better than the other and how system/subsystem parameters should be selected in order to operate efficiently.

### 1.1.2 Schedulability analysis

Supporting global shared resource among subsystems is a major challenge as it increases the complexity of the system analysis considerably. Due to this complexity, the schedulability analysis of both SIRAP and *Overrun* (also HSRP) are based on some simplifying assumptions which make them easier. The consequence of these simplifying assumptions is that the analysis may become very pessimistic, potentially requiring more CPU resources than what is actually needed. Therefore we aim at reducing the potential pessimism in the schedulability analysis of SIRAP and HSRP by introducing tighter analysis.

- In paper D we show that the schedulability analysis associated with the SIRAP protocol can be pessimistic if the number of shared resources and/or number of resource accesses is high. We present two different schedulability analysis approaches for SIRAP. The results obtained from simulation analysis show that the new approaches can decrease the CPU resources allocated to each subsystem significantly compared with the original schedulability analysis.

- In paper E we show that the existing schedulability analysis of the *Overrun* (without payback) is pessimistic\(^1\). We present a tighter analysis that reduce the required CPU resource demand. In addition we evaluate the improvements that the new analysis can achieve compared with the traditional analysis. Depending on the system parameters, a significant improvement in the CPU resource usage can be achieved when using the new analysis. However, the time complexity of the new analysis is higher than the existing analysis presented in paper B.

\(^1\)The pessimism in the schedulability analysis is also included in the original analysis of HSRP.
1.1.3 Algorithms for efficient CPU resource usage

It is required that the HSF should use the CPU-resources efficiently, i.e., given a particular system configuration, the system load should be minimized. However, it may not be straightforward to find the optimal subsystems parameters that generate the minimum system load without violating the requirement of independent subsystem development, i.e., without knowledge about temporal behavior of other subsystems that will be integrated on the same CPU. By taking this contradiction between allowing for independent development of subsystems, and minimizing system load, into account, we propose approaches and algorithms that can decrease the CPU resources demand.

- For SIRAP, we show that it is possible to reduce the allocated CPU resource needs for a subsystem by manipulating the ceiling of resources in paper F. Based on this, we propose an algorithm that selects the optimal resource ceiling value per global shared resource that will be used during self-blocking, resulting in the lowest CPU resources allocation needs for that subsystem.

- For the Overrun, and considering the requirement of subsystem independent development, we propose a two-step approach to find an optimal solution to the system load minimization problem in paper G. In the first step, and for each subsystem in isolation, an algorithm is proposed to derive a set of interface candidates. In the second step, during system integration, another algorithm is used to select one candidate for each subsystem that minimizes the system load.

1.2 Outline of thesis

The outline of this thesis is as follows: in Chapter 2 we explain and define the basic concepts of real-time systems, and the terms that will be used throughout this thesis. In Chapter 3 we describe the hierarchical scheduling framework, we address the problem of allowing global shared resource between subsystems and we present some solutions for this problem. In Chapter 4 we present our conclusion and suggestions for future work. We present the technical overview of the papers that are included in this thesis in Chapter 5 and we present these papers in Chapters 6-12.
Chapter 2

Background and System Model

In this chapter we present some basic concepts concerning real-time systems, as well as the system model that will be used in the next chapters.

2.1 Real-time systems

A real-time system is a computing system whose correctness relies not only on its functionality, but also on timeliness, i.e., the system should produce correct results at correct instances of time [6]. Real-time systems are usually constructed using concurrent programs called tasks and each task is supposed to perform a certain functionality (for example reading a sensor value, computing output values, sending output values to other tasks or devices, etc). A real-time task should complete its execution before a predefined time called deadline.

Real-time tasks can be classified according to their timing constraint to either hard real-time tasks or soft real-time tasks. For hard real-time tasks, all tasks should complete their execution before their deadlines otherwise a catastrophic consequence may occur. However, for soft real-time tasks, it is acceptable that deadlines are missed which may degrade the system performance, e.g., in a mobile phone where missing some deadlines will decrease the quality of the sound. Many systems contain a mix of hard and soft real-time tasks.

A real-time task consists of an infinite sequence of activities called jobs, and depending on their execution behaviors, real-time tasks are modeled as...
Chapter 2

Background and System Model

In this chapter we present some basic concepts concerning real-time systems, as well as the system model that will be used in the next chapters.

2.1 Real-time systems

A real-time system is a computing system whose correctness relies not only on its functionality, but also on timeliness, i.e., the system should produce correct results at correct instances of time [6]. Real-time systems are usually constructed using concurrent programs called tasks and each task is supposed to perform a certain functionality (for example reading a sensor value, computing output values, sending output values to other tasks or devices, etc). A real-time task should complete its execution before a predefined time called deadline.

Real-time tasks can be classified according to their timing constraint to either hard real-time tasks or soft real-time tasks. For hard real-time tasks, all tasks should complete their execution before their deadlines otherwise a catastrophic consequence may occur. However, for soft real-time tasks, it is acceptable that deadlines are missed which may degrade the system performance, e.g., in a mobile phone where missing some deadlines will decrease the quality of the sound. Many systems contain a mix of hard and soft real-time tasks.

A real-time task consists of an infinite sequence of activities called jobs, and depending on their execution behaviors, real-time tasks are modeled as
either periodic, sporadic or aperiodic tasks:

- Periodic tasks have a fixed inter-arrival time called period, i.e., a periodic task become ready to execute every predefined period.

- Sporadic tasks have known minimum inter-arrival time while the maximum inter-arrival time can be infinity.

- Aperiodic tasks are triggered at arbitrary times, with no known minimum inter-arrival time.

2.1.1 Scheduling algorithms

In a single processor, the CPU can not be assigned to more than one task at any given point in time. If a set of tasks are ready to execute, then a scheduling criterion should be used to define the execution order of these tasks. The scheduling criterion uses a set of rules defined by a scheduling algorithm to determine the execution order of the task set. If all tasks complete their execution before their deadlines then the schedule is called a feasible schedule and the tasks are said to be schedulable. If the scheduler permit other tasks to interrupt the execution of the running task (task in execution) before completing of its execution then the scheduling algorithm is called preemptive, otherwise it is called a non-preemptive scheduling algorithm.

Real-time scheduling falls in two basic categories; online scheduling (the order of task execution is determined during runtime) and offline scheduling (a schedule is created before runtime)[7].

For online scheduling, the order of task execution is determined during runtime according to task priorities. The priorities of tasks can be static which means that the priorities of tasks will not change during runtime. This type of scheduling algorithm is called Fixed Priority Scheduling (FPS) and both Rate Monotonic (RM) scheduling [8] and Deadline Monotonic (DM) [9] are examples of this type of scheduling. In RM, the priorities of the tasks are assigned according to their periods, while for DM the priority of a task is assigned based on its deadlines. The task priorities can be dynamic which means that they can change during runtime, and Earliest Deadline First (EDF) [8] is an example of such a scheduler. For EDF, the task that has earlier deadline among all ready tasks will execute first.
2.1.2 Logical resource sharing

A logical resource is any software structure that can be used by a task to advance its execution [10]. For example a resource can be a data structure, flash memory, a memory map of a peripheral device. If more than one task uses the same resource, then that resource is called shared resource. The part of task’s code that uses a shared resource is called critical section. When a job enters a critical section (starts accessing a shared resource) then no other jobs, including the jobs of higher priority tasks, can access the shared resource until the accessing job exits the critical section (mutual exclusion method). The reason is to guarantee the consistency of the data in the shared resource and this type of shared resource is called non-preemptable resource. For preemptive scheduling algorithms, sharing logical resources cause a problem called priority inversion [4]. The priority inversion problem happens when a job with high priority must access a shared resource that is currently accessed by another lower priority job, so the higher priority job will not be able to preempt the lower priority job. The higher priority job will be blocked until the lower priority job releases the shared resource. The time that the high priority job will be blocked can be unbounded since other jobs with intermediate priority that do not access the shared resource can preempt the low priority job while it is executing inside its critical section. As a result of the priority inversion problem, the higher priority job may miss its deadline. A proper protocol should be used to synchronize the access to the shared resource in order to bound the waiting time of the blocked tasks. Several synchronization protocols, such as the Priority Inheritance Protocol (PIP) [3], the Priority Ceiling Protocol (PCP) [4] and the Stack Resource Policy (SRP) [5], have been proposed to solve the problem of priority inversion. We will explain the SRP protocol in details, a protocol central for this thesis, suitable for RM, DM, and EDF scheduling algorithms.

Stack resource policy To describe how SRP works, we first define some terms that are used with SRP.

- **Preemption level.** Each task has a preemption level which is a static value proportional to the inverse of task relative deadline for the EDF scheduling. For RM/DM the preemption level equals to the priority of the task.

- **Resource ceiling.** Each shared resource is associated with a resource ceiling which equals to the highest preemption level of all tasks that use the resource.
• **System ceiling.** System ceiling is a dynamic parameter that changes during execution. The system ceiling is equal to the currently locked highest resource ceiling in the system. If at any time there is no accessed shared resource then the system ceiling would be equal to zero.

According to SRP, a job \( j_i \) generated by task \( \tau_i \) can preempt the currently executing job \( j_k \) only if \( j_i \) is a higher-priority job of \( j_k \) and the preemption level of \( \tau_i \) is greater than the current subsystem ceiling.

### 2.2 System model

In this thesis, our focus is on a two-level hierarchical scheduling framework where a system \( S \), executing on a single processor, consists of one or more subsystems \( S_s \in S \). The hierarchical scheduling framework can be generally represented as a two-level tree of nodes, where each node represents a subsystem with its own scheduler for scheduling internal tasks, and CPU time is allocated from a parent node to its children nodes, as illustrated in Figure 2.1. Each subsystem \( S_s \) consists of a set of tasks and a scheduler as shown in Figure 2.1. During runtime, the system level scheduler (global scheduler) selects which subsystem will access the CPU resources. Once a subsystem is assigned the processor, the corresponding subsystem scheduler (local scheduler) selects which task that will be executed.

In this thesis, tasks from different subsystems are allowed to access logical shared resources. Let \( R_s \) denote the global shared resources accessed by \( S_s \). Let us also define resource holding time \( X_{sk} \) as the maximum time that a task of \( S_s \) may lock a resource \( R_k \in R_s \). Finally, let \( X_s = \{ X_{sk} \} \) denote the set of maximum resource holding times \( X_s = \{ X_{sk} \} | \forall R_k \in R_s \).

Shin and Lee [2] proposed the notion of subsystem timing interface that abstract the collective temporal requirements of each subsystem, based on the periodic resource model and assuming that tasks do not share global shared resources. The periodic resource model \( \Gamma_s(P_s, Q_s) \) includes \( P_s \) as a subsystem period and \( Q_s \) as a subsystem budget (which represents a periodic CPU resources allocation time). To consider the problem of global shared resources in the HSF, we extend the subsystem timing interface by including a third parameter on it, i.e., a resource holding time \( X_s \). Note that, we assume that each subsystem is associated with a subsystem timing interface and it is used to perform the composability test of the subsystems. Moreover, the subsystem timing interface is used by the global scheduler during runtime, to assign CPU resources to the subsystem.
For the task model, we consider a deadline-constrained sporadic hard real-time task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$ where $T_i$ is the minimum separation time between its successive jobs, $C_i$ is the worst-case execution time, and $D_i$ is a relative deadline ($C_i \leq D_i \leq T_i$). Each task is allowed to access one or more logical shared resources and each element $c_{i,j} \in \{c_{i,j}\}$ is a critical section length that represents the worst-case execution time of $\tau_i$’s access to a global shared resource $R_j$.

In this thesis, we assume that both local and global schedulers use the fixed priority preemptive scheduling policy (FPS), however, most of the results of this thesis are not limited to FPS and can be extended to include other paradigms such as EDF (papers A and B present analysis for EDF as well). Additionally, we focus only on two synchronization protocols that handle the problem of sharing global shared resources, i.e., SIRAP and Overrun (without payback), as these protocols have similarities in their analysis (use the periodic resource model) and implementation (use the periodic server\(^1\)). Finally, the SRP protocol is assumed to be used within a subsystem to arbitrate the access of shared resources by tasks.

Note that throughout the seven included papers in this thesis, there are some differences in notations, indexes and terminologies. Therefore it is important to read and follow the corresponding system model of each paper, respectively.

---

\(^1\)A periodic server is a server that executes as a periodic task.
Chapter 3

A Real-Time Hierarchical Scheduling Framework with Logical Resource Sharing

In this chapter, we will first describe the analysis of the HSF. Then we will explain the problem of accessing global shared resources in a hierarchical scheduling framework, and we present some protocols that can handle this problem.

Over the years, there has been a growing attention to using hierarchical scheduling for real-time systems. Deng and Liu [11] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [12] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [13, 14] presented schedulability analysis techniques for the EDF-based global schedulers. Mok et al. [15, 16] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [2] introduced the periodic virtual processor model (to characterize the periodic CPU allocation behaviour), and many studies have been proposed on schedulability analysis with this model under fixed-priority scheduling [17, 18, 19] and under EDF scheduling [2, 20]. Being central to this thesis, the virtual periodic resource model is presented in detail in this chapter. Easwaran et al. [21] introduced the Explicit Deadline Periodic (EDP) virtual processor model.
Chapter 3

A Real-Time Hierarchical Scheduling Framework with Logical Resource Sharing

In this chapter, we will first describe the analysis of the HSF. Then we will explain the problem of accessing global shared resources in a hierarchical scheduling framework, and we present some protocols that can handle this problem.

Over the years, there has been a growing attention to using hierarchical scheduling for real-time systems. Deng and Liu [11] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [12] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [13, 14] presented schedulability analysis techniques for the EDF-based global schedulers. Mok et al. [15, 16] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [2] introduced the periodic virtual processor model (to characterize the periodic CPU allocation behaviour), and many studies have been proposed on schedulability analysis with this model under fixed-priority scheduling [17, 18, 19] and under EDF scheduling [2, 20]. Being central to this thesis, the virtual periodic resource model is presented in detail in this chapter. Easwaran et al. [21] introduced the Explicit Deadline Periodic (EDP) virtual processor model. A
common assumption shared by all above studies is that tasks are independent, i.e., tasks are not allowed to share logical resources.

In this thesis, we relax the assumption of independent tasks by addressing the challenge of enabling efficient compositional integration for independently developed subsystems interacting through sharing of global shared resources. In particular, we extend the HSF proposed by Shin and Lee \[2\] enabling sharing of global shared resources.

### 3.1 HSF schedulability analysis

In the following sections, we will explain how to evaluate the subsystem timing interface and we will show how to verify the composability of the system. We will first explain the virtual processor resource model and the local schedulability analysis that are used to evaluate the subsystem timing interface; the subsystem’s abstract notion of resource requirement needed to ensure correct timing.

#### 3.1.1 Virtual processor model

The notion of real-time virtual processor (resource) model was first introduced by Mok \textit{et al.} \[15\] to characterize the CPU allocations that a parent node provides to a child node in a hierarchical scheduling framework. The \textit{CPU supply} of a virtual processor model refers to the amount of CPU allocations that the virtual processor model can provide. The \textit{supply bound function} of a virtual processor model calculates the minimum possible CPU supply of the virtual processor model for a time interval length $t$.

Shin and Lee \[2\] proposed the periodic virtual processor model $\Gamma(P_s, Q_s)$, where $P_s$ is a subsystem period ($P_s > 0$) and $Q_s$ is a subsystem budget ($0 < Q \leq P$). The supply bound function $\text{sbf}_s(t)$ (shown in Figure 3.1) of the periodic resource model is computed as follows;

$$\text{sbf}_s(t) = \begin{cases} t - (k+1)(P_s - Q_s) / (j-1)Q_s & \text{if } t \in V^k \\ \text{otherwise} \end{cases}$$

where $k = \max(\lceil (t - (P_s - Q_s)) / P_s \rceil, 1)$ and $V^k$ denotes an interval $[(k+1)P_s - 2Q_s, (k+1)P_s - Q_s]$ in which the subsystem $S_s$ receives budget. To guarantee a minimum CPU resource supply, the worst-case budget provision is considered in Eq. (3.1) assuming that 1) tasks are released at the same
Chapter 3. A Real-Time Hierarchical Scheduling Framework

3.1 HSF schedulability analysis

The notion of real-time virtual processor (resource) model was first introduced by Mok et al. [15] to characterize the CPU allocations that a parent node provides to a child node in a hierarchical scheduling framework. The CPU supply of a virtual processor model refers to the amount of CPU allocations that the virtual processor model can provide. The supply bound function of a virtual processor model calculates the minimum possible CPU supply of the virtual processor model for a time interval length $t$.

Shin and Lee [2] proposed the periodic virtual processor model $\Gamma(P_s, Q_s)$, where $P_s$ is a subsystem period ($P_s > 0$) and $Q_s$ is a subsystem budget ($0 < Q_s \leq P_s$). The supply bound function $sbf(t)$ (shown in Figure 3.1) of the periodic resource model is computed as follows:

$$sbf(t) = \begin{cases} t - (k+1)(P_s - Q_s) & \text{if } t \in V_k(j-1)Q_s \\ 0 & \text{otherwise} \end{cases},$$

where $k = \max(\lceil \frac{t - (P_s - Q_s)}{P_s} \rceil, 1)$ and $V_k$ denotes an interval $[(k+1)P_s - 2Q_s, (k+1)P_s - Q_s]$ in which the subsystem $S_s$ receives budget.

To guarantee a minimum CPU resource supply, the worst-case budget provision is considered in Eq. (3.1) assuming that I) tasks are released at the same time when the subsystem budget depletes (at time $t = 0$ in Figure 3.1), II) the first budget was supplied at the beginning of the period, and III) all following budgets will be supplied at then end of the subsystem period. By considering the worst-case budget provision, a subsystem assumes that the maximum interference from higher priority subsystems will occurs which delays the budget supply to the end of the subsystem period, and by this it does not require any information from other subsystems that will be integrated. Note that, depending on the parameters of the subsystem, the assumption of the worst-case budget provision may make the analysis pessimistic (see Section 6.6.5).

![Figure 3.1: The supply bound function of a periodic virtual processor model $\Gamma(P, Q)$ for $k = 3$.](image)

3.1.2 Local schedulability analysis

The local schedulability analysis is used to check whether a given periodic resource model $\Gamma(P, Q)$ (subsystem period and budget) can guarantee the temporal requirements of the subsystem internal tasks. The local schedulability analysis is as follows [2]:

$$\forall \tau_i, 0 < \exists t \leq D_i \quad rbf_{FP}(i, t) \leq sbf(t),$$

(3.2)
where \( \text{rbf}_{FP}(i, t) \) denotes the \textit{request bound function} of a task \( \tau_i \) which computes the maximum cumulative execution requests that could be generated from the time that \( \tau_i \) is released up to time \( t \). \( \text{rbf}_{FP}(i, t) \) is computed as follows

\[
\text{rbf}_{FP}(i, t) = b_i + C_i + \sum_{\tau_k \in \text{hp}(i)} \left\lfloor \frac{t}{T_k} \right\rfloor \cdot C_k,
\]

where \( \text{hp}(i) \) is the set of tasks with priorities higher than \( \tau_i \) and \( b_i \) is the maximum blocking time from lower priority tasks.

### 3.1.3 System composability

For a system \( S \) that consists of a set of \( m \) subsystems \( S_1, S_2, \ldots, S_m \), the subsystems are composable if each subsystem receives sufficient execution satisfying its timing interface [2], in other words, the subsystems are schedulable under the given global (system level) scheduler. As long as the periodic resource model is used in evaluating the subsystem timing interface, each subsystem can be modeled as a simple periodic task where the subsystem period is equivalent to the task period and the subsystem budget is equivalent to the worst case execution time of a task. Moreover, resource holding times can be modeled as critical section execution times. Then the schedulability analysis used for simple periodic tasks with resource sharing can be used. Hence, the general condition for global schedulability is,

\[
\forall S_s \exists t : 0 < t \leq P_s, \quad \text{RBF}_s(t) \leq t,
\]

where \( \text{RBF}_s(t) \) denotes the request bound function of a subsystem \( S_s \) and it is computed as follows,

\[
\text{RBF}_s(t) = Q_s + B_s + \sum_{S_k \in \text{hp}(s)} \left\lfloor \frac{t}{F_k} \right\rfloor \cdot Q_k,
\]

where \( \text{hp}(i) \) is the set of subsystems with priorities higher than that of \( S_s \) and \( B_s \) is the maximum blocking time (resource holding time) imposed to a subsystem \( S_s \), when it is blocked by lower-priority subsystems (suppose that \( S_j \) imposes the maximum blocking on \( S_s \) then \( B_s = X_j \)).

### 3.1.4 Subsystem interface evaluation

Looking at Eq. 3.5, the composability of a system is increased when decreasing the subsystem utilization \( Q_s/P_s \). Finding optimal values for \( P_s \) and \( Q_s \) that
3.2 Global resource sharing

In this section we will focus on the problem of supporting global shared resources such that tasks from different subsystems share logical resources. The logical resources that are shared by tasks from the same subsystem are called local shared resources and the works presented in [12, 17, 25] show that using existing synchronization protocols such as SRP can handle the problem of sharing local resources without any modification. However, for global shared resources, new synchronization protocols are required. First we explain the problem of supporting global shared resources followed by discussing some solutions.
3.2.1 Problem formulation

When a task $\tau_j$ locks a shared resource $R_k$, all other tasks that want to access the same resource $R_k$ will be blocked until $\tau_j$ releases it. To achieve a predictable real-time behaviour, the waiting time of other tasks that want to access a locked shared resource should be bounded. The traditional synchronization protocols, such as SRP used with non-hierarchical scheduling, can not without modification handle the problem of sharing global shared resources in a hierarchical scheduling framework. To explain the reason, suppose $\tau_j$ that belongs to a subsystem $S_I$ is holding a logical resource $R_1$, the execution of the task $\tau_j$ can be preempted while $\tau_j$ is executing inside the critical section of the resource $R_1$ (see Fig 3.2) due to the following reasons:

1. **Intra subsystem preemption**, a higher priority task $\tau_k$ within the same subsystem preempts the task $\tau_j$.

2. **Inter subsystem preemption**, a ready task $\tau_c$ that belongs to a subsystem $S_P$ preempts $\tau_j$ when the priority of subsystem $S_P$ is higher than the priority of subsystem $S_I$.

3. **Budget depletion inside a critical section**, if the budget of the subsystem $S_I$ depletes, the task $\tau_j$ will not be allowed to execute until the budget of its subsystem will be replenished at the beginning of the next subsystem period $P_I$.

![Figure 3.2: Task preemption while running inside a critical section.](image)
The SRP protocol can only solve the problem caused by task preemption within a subsystem (case number 1) since there is a direct relationship between the priorities of tasks within the same subsystem. Moreover, if tasks are from different subsystems (inter task preemption) then priorities of tasks belonging to different subsystems are independent. Still, the priorities of the subsystems can be used to solve this problem, and SRP can be used between subsystems such that, if a task that belongs to a subsystem locks a global shared resource, then this subsystem blocks all other subsystems if any of their internal tasks want to access the same global shared resource. However, SRP can not handle case number 3, i.e., budget depletion inside a critical section. Budget depletion can cause a problem if it happens while a task $\tau_j$ of a subsystem $S_I$ is executing inside the critical section of a global shared resource $R_1$. If another task $\tau_m$, belonging to another subsystem, is waiting for the same resource $R_1$, this task must wait until $S_I$ is replenished so $\tau_j$ can continue executing until it releases the lock on resource $R_1$. This waiting time exposed to $\tau_m$ can be potentially very long, causing $\tau_m$ to miss its deadline.

### 3.3 Supporting global resource sharing

Four different mechanisms have been proposed to enable resource sharing in the context of hierarchical scheduling. The mechanisms use different methods to bound the waiting time of tasks that share the same global shared resources. Three of them use the SRP protocol to synchronize access to a global shared resource, while the forth mechanism uses an extended version of PIP.

If SRP is used in a HSF then the SRP’s associated terms resource and system ceiling should be extended as follows:

**Resource ceiling**: With each global shared resource $R_k$, two types of resource ceilings are associated; an internal resource ceiling ($rc_{sk}$) for local scheduling and an external resource ceiling ($RX_k$) for system level scheduling.

**System/subsystem ceiling**: The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling (i.e. highest priority) of a currently locked resource in the system/subsystem.

Solving the problem of budget depletion inside a critical section can be done following one of the three approaches:

- Preventing a task from locking a shared resource if its subsystem does not has enough remaining budget (skipping and deadline shifting mechanisms).
• Adding extra resources to the budget of each subsystem to prevent the budget depletion inside a critical section (overrun mechanism).

• Using the budget of other subsystems when their internal tasks want to access an already locked global shared resource (bandwidth inheritance).

The following sections explain different synchronization protocols using these three approaches.

3.3.1 SIRAP

The Subsystem Integration and Resource Allocation Policy (SIRAP) [26] protocol is based on the skipping mechanism to prevent depletion of the budget during global shared resource access. SIRAP uses the periodic resource model and its mechanism works as follows; when a task $\tau_i$ tries to access a global shared resource $R_k$, SIRAP checks the remaining budget before granting the access to the global shared resource; if there is sufficient remaining budget to lock and release $R_k$ before budget depletion (i.e., the currently un-consumed budget is greater than the maximum time that $\tau_i$ may lock $R_k$), then the task enters the critical section, and it updates both system and subsystem ceiling. If there is insufficient remaining budget, SIRAP takes the following actions:

• **Self-blocking**: the job of $\tau_i$ is blocked until the next following budget replenishment so, at that time, there will be enough budget to lock and release the shared resource $R_k$.

• **Subsystem ceiling**: the subsystem ceiling will be updated to bound the minimum required subsystem budget, while the system ceiling is only updated when a global shared resource is accessed so that tasks from other subsystems can access the global shared resource $R_k$.

• **Budget replenishment**: at the time instant when the subsystem budget is replenished, the state of the job of task $\tau_i$ will be changed to ready such that it can execute and lock the global shared resource.

The reason of updating the subsystem ceiling during the self-blocking is to bound the minimum required budget. If tasks with priority lower than that of $\tau_i$ are allowed to execute while $\tau_i$ is in the self-blocking state, it may cause additional self-blocking. This is unpractical as the subsystem budget should be big enough to finish all self-blocking within one budget supply. In paper F we have proposed an algorithm that finds the best value of subsystem ceiling
during the self-blocking state in order to decrease the minimum subsystem budget.

Figure 3.3 illustrates an example of a self-blocking occurrence during the execution of subsystem $S_s$. A job of a task $\tau_i$ tries to lock a global shared resource $R_k$ at time $t_2$. It first determines the remaining subsystem budget $Q^r$ (which is equal to $Q^r = Q_s - (Q^1 + Q^2)$, i.e., the subsystem budget left after consuming $Q^1 + Q^2$). Next, it checks if the remaining budget $Q^r$ is greater than or equal to the maximum resource locking time ($X_{ik}$) of the job access to $R_k$, i.e., if ($Q^r \geq X_{ik}$). In Figure 3.3, this condition is not satisfied, so $\tau_i$ blocks itself and is not allowed to execute before the next replenishment period ($t_3$ in Figure 3.3).

![Figure 3.3: An example illustrating self-blocking.](image)

SIRAP uses the periodic resource model to abstract the timing requirements of each subsystem. The effect of using SIRAP on the schedulability analysis appears in the local schedulability analysis in Eq. (3.2) either on $rbf_{FP}(i, t)$ or $sbf(t)$ (papers A and D).
3.3.2 HSRP

The Hierarchical Stack Resource Policy (HSRP) [25] extends the SRP protocol to handle the sharing of global shared resources problem in hierarchical scheduling frameworks. HSRP is based on the overrun mechanism and it works as follows: when the budget of a subsystem expires and a job of task $\tau_i$ that belong to the subsystem has not released the lock of a global shared resource, the subsystem overruns its budget and the job continues its execution until it releases the locked resource. When a job accesses a global shared resource its priority is increased to the highest local priority to prevent any preemption from other tasks that belong to the same subsystem during the access of the shared resource. SRP is used at the global level to synchronize the execution of subsystems. Each global shared resource has a ceiling equal to the maximum priority of the subsystem that its internal tasks may access that resource. Two versions of the overrun mechanism have been presented: 1) The overrun mechanism with payback which works as follows: whenever overrun happens in a subsystem, the budget of the subsystem will be decreased by the amount of the overrun time in its next execution instant as shown in Figure 3.4a. 2) In the second version which is called overrun mechanism without payback, no further actions will be taken after the event of an overrun (see Figure 3.4b). Selecting which of these two mechanisms gives better results, in terms of task response times depends on the system parameters. The presented schedulability analysis does not support composability, disallowing independent analysis of individual subsystems, since information about other subsystems is needed in order to apply the schedulability analysis for all tasks. In paper B the analysis of HSRP has been extended to support compositional scheduling based on the periodic resource model. In addition, and to generalize the local analysis, SRP is assumed to be used locally and globally. The effect of using Overrun on the schedulability analysis is added to the global schedulability analysis in Eq. (3.5).

3.3.3 The BROE server

The Bounded-delay Resource Open Environment (BROE) server [29] extends the Constant Bandwidth Server (CBS) [30] in order to handle the problem of sharing logical resources in a hierarchical scheduling framework. The analysis associated with the BROE server supports independently developed subsystems (open environment). BROE uses the bounded-delay resource model [15] to characterize the CPU allocations for each subsystem. Because of using the
3.3 Supporting global resource sharing

3.3.2 HSRP

The Hierarchical Stack Resource Policy (HSRP) [25] extends the SRP protocol to handle the sharing of global shared resources problem in hierarchical scheduling frameworks. HSRP is based on the overrun mechanism and it works as follows: when the budget of a subsystem expires and a job of task $\tau_i$ that belongs to the subsystem has not released the lock of a global shared resource, the subsystem overruns its budget and the job continues its execution until it releases the locked resource. When a job accesses a global shared resource, its priority is increased to the highest local priority to prevent any preemption from other tasks that belong to the same subsystem during the access of the shared resource. SRP is used at the global level to synchronize the execution of subsystems. Each global shared resource has a ceiling equal to the maximum priority of the subsystem that its internal tasks may access that resource.

Two versions of the overrun mechanism have been presented: 1) The overrun mechanism with payback which works as follows: whenever overrun happens in a subsystem, the budget of the subsystem will be decreased by the amount of the overrun time in its next execution instant as shown in Figure 3.4a. 2) In the second version which is called overrun mechanism without payback, no further actions will be taken after the event of an overrun (see Figure 3.4b). Selecting which of these two mechanisms gives better results, in terms of task response times depends on the system parameters. The presented schedulability analysis does not support composability, disallowing independent analysis of individual subsystems, since information about other subsystems is needed in order to apply the schedulability analysis for all tasks. In paper B the analysis of HSRP has been extended to support compositional scheduling based on the periodic resource model. In addition, and to generalize the local analysis, SRP is assumed to be used locally and globally. The effect of using Overrun on the schedulability analysis is added to the global schedulability analysis in Eq. (3.5).

3.3.3 The BROE server

The Bounded-delay Resource Open Environment (BROE) server [29] extends the Constant Bandwidth Server (CBS) [30] in order to handle the problem of sharing logical resources in a hierarchical scheduling framework. The analysis associated with the BROE server supports independently developed subsystems (open environment). BROE uses the bounded-delay resource model [15] to characterize the CPU allocations for each subsystem. Because of using the bounded-delay resource model, the timing interface of a subsystem includes server speed, delay tolerance and resource holding time. These parameters are used both during design time, to verify the composability of the system (global schedulability analysis), as well as during runtime, when the global scheduler allocates the required CPU for each subsystem. The BROE server is based on the deadline shifting mechanism and it uses the SRP protocol to arbitrate access to global shared resources. In order to prevent budget depletion inside critical sections, the subsystem performs a budget check before accessing a global shared resource. If the subsystem remaining budget is sufficient, it allows the task to lock the global shared resource. Otherwise it postpones its current deadline and replenishes its budget (according to certain rules that guarantee the correctness of the CBS servers execution and the CPU resource model supply) to be able to lock and release the global shared resource safely. Finally, as BROE extends the CBS server, only the EDF scheduling algorithm can be used in the global (system level) scheduling.
3.3.4 BWI

The BandWidth Inheritance protocol (BWI) [31] extends the resource reservation framework to systems where tasks can share resources. The BWI approach uses (but is not limited to) the CBS algorithm together with a technique that is derived from the Priority Inheritance Protocol (PIP). According to BWI, each task is scheduled through a server, and when a task that is executing inside a lower priority server blocks another task executing in a higher priority server, the blocking task will be added to the higher priority server. When the task releases the shared resource, it will be discarded from the high priority server. For schedulability analysis, each server should include a characterization of interference time due to adding lower priority tasks in the server. This approach is suitable for systems where the execution time of a task inside a critical section cannot be predicted. In addition, the scheduling algorithm does not require any a-prior knowledge about which shared resources that tasks will access nor the arrival time of tasks. However, BWI is not suitable for systems that consist of many hard real-time tasks. The reason is that the worst-case interference from all tasks that belong to other servers and access global shared resources will be added to the budget of each server to guarantee the timing requirements of the tasks. Hence, BWI becomes pessimistic in terms of CPU-resource usage for systems that have hard real-time tasks accessing global shared resources. In this thesis we consider systems with hard real-time requirements, hence we will exclude further discussion of BWI.

3.4 Isolation between subsystems

One of the key advantages of hierarchical scheduling is that it provides an isolation between subsystems during runtime, i.e., the execution of tasks that belong to a subsystem will not affect the execution of tasks that belong to other subsystems in an unpredictable manner. A full isolation between subsystems can be achieved for independent subsystems where tasks do not share global shared resources. However, sharing of global shared resources makes it more difficult to guarantee isolation between subsystems during runtime. There are two reasons that may violate the isolation between subsystems:

- The execution time of a job inside a critical section may exceed its estimated worst-case execution time. This may increase the resource holding time and may affect the global schedulability of the system. As a result, the other subsystems may not get enough CPU resources and their
tasks may miss their deadlines. To solve this problem, a runtime mecha-
nism should be used to make sure that a task will never exceed its critical 
section execution time when it accesses a global shared resource.

- A task may corrupt the data of a global shared resource which may af-
  fect all subsystems that share the same global shared resource [29]. If 
  the corrupted data is detectable by the system then a backup strategy may 
  be used to restore the latest correct data. Otherwise it is impossible to 
  achieve a full isolation between subsystems and it will be the responsi-
  bility of the designer/developer to avoid the use of such global shared 
  resources.

3.5 Comparing SIRAP, HSRP and BROE

This section compares HSRP, BROE, and SIRAP, looking at some theoretical 
properties, implementation complexity and overhead.

3.5.1 Theoretical comparison

A detailed systematic comparison may not be possible/fair between the three 
protocols SIRAP, HSRP and BROE, since each protocol has different assump-
tions, settings and goals. For example, in both HSRP and BROE it is assumed 
that subsystem parameters (period and budget) are given and it is required to 
verify the schedulability of the system, while the goal of the analysis associated 
with SIRAP is to find optimal/suboptimal subsystem parameters that increase 
the composability of the system. In addition, the schedulability algorithms 
used in the three approaches are different; HSRP uses only FPS in both sub-
system and system level, while BROE uses only EDF and SIRAP uses either 
FPS or EDF in the local and global level. Another difference between the pro-
tocols is the resource supply model; BROE uses the bounded-delay resource 
model while SIRAP uses the periodic resource model, and HSRP uses the pe-
riodic resource model implicitly. Furthermore, HSRP uses SRP in the global 
level while locally it uses a simple non-preemptive approach when a task ac-
cesses a global shared resource. For BROE and SIRAP, SRP is used locally 
and can also be used globally depending on the required level of abstraction 
of the subsystem timing interface. Finally, the analysis associated with HSRP 
does not support independent subsystem development, while this is supported 
by SIRAP and BROE.
In this thesis, we aim at defining a common framework in which we can compare the mechanisms used by the protocols, such that we can compare the protocols and/or properties of the protocols.

One of the properties that can be used to compare efficiency of the protocols is the system load, which is a measure of the amount of collective CPU requirements necessary to guarantee the schedulability of an entire framework. By minimizing the system load, more subsystems can be integrated in a single processor, which makes the framework cost-efficient and applicable for a wider range of applications.

In spite of the differences between the three protocols, a theoretical high level comparison can be carried out based on their schedulability analysis. Comparing the schedulability analysis of SIRAP and HSRP, it is not possible to prove that one of the protocols is more efficient than the other. The reason is that their efficiency depends on the subsystem parameters as well as on the parameters of the shared resources (even between the two types of overrun mechanisms presented in [25] is not easy to find which of them that requires less system load). In paper C we have compared SIRAP and Overrun by means of simulation analysis using the same assumptions and settings. The results of this comparison confirmed our conclusion that the efficiency of the mechanisms depends exclusively on the system parameters.

BROE seems to be more efficient than the other two as it does not add direct effect on the local and global schedulability analysis. However, BROE uses the bounded-delay resource model which may affect both the local and global schedulability analysis compared with the periodic resource model used by SIRAP and HSRP. The periodic resource model provides more CPU resources than the bounded-delay resource model. As a result, it may affect the local schedulability analysis and it may require larger subsystem budget compared with the case of using periodic resource model. In the global schedulability analysis, because of using the bounded-delay resource model, it is only possible to use an approximated schedulability analysis with BROE, while for the other protocols, exact schedulability analyses can be applied which give tighter (less pessimistic) results. Considering this effect, there can be some cases when SIRAP or HSRP give better results than BROE and of course the performance also depends on the system parameters.

3.5.2 Implementation complexity and overhead

Both SIRAP and HSRP rely on using the periodic server to implement each subsystem, and a server is assigned $Q_s$ budget every period $P_s$. Implementing
the BROE server is done relying on an EDF global scheduler together with a modified version of CBS. Comparing the two types of servers, the implementation of the periodic server is easier and has less runtime overhead than the implementation of the CBS server (CBS has more states). Comparing the SIRAP and HSRP implementations using the periodic server, the implementation of HSRP impose more runtime overhead than SIRAP as it requires to change the behavior of the global scheduler when an overrun occurs. SIRAP does not require any change in the execution of the periodic server during runtime. An implementation of both SIRAP and HSRP has been presented in [32, 33], and the results show that the primitives that are used to implement HSRP impose more runtime overhead than using the SIRAP primitives. In addition, the number of scheduler calls will be higher for HSRP than SIRAP, which increases the runtime overhead. In [34] the implementation of all three protocols including BROE is presented and the results show that BROE imposes the highest runtime overhead compared with the other protocols to solve the problem of budget depletion. The reason for this is that BROE may change the deadline and the replenishment time of the server more often and such operations are relatively expensive, in terms of runtime overhead, as they require to arrange the ready queue of the server. On the other hand, SIRAP requires that the resource holding times of all tasks accessing any global shared resource should be provided during runtime, while the other two protocols may require the maximum resource locking time for each subsystem. This may require more memory space storing the resource holding times, which is one of the disadvantages of SIRAP compared with the other protocols (HSRP and BROE). One way to decrease the memory space for SIRAP is to consider the maximum value of resource holding times per global shared resource or among all global shared resources. However, this should be taken into account in the local schedulability analysis which makes the results of SIRAP less accurate (more pessimistic).
Chapter 4

Summary, Conclusions and Future Work

In this thesis we have addressed the problem of supporting global shared resources in the context of hierarchical scheduling. For this purpose, we have presented a novel synchronization protocol called Subsystem Integration and Resource Allocation Policy (SIRAP), which provides temporal isolation between subsystems that share logical resources. We have evaluated the overhead introduced by SIRAP through a simulation study. To decrease this overhead, the results of the study showed that the subsystem period should be chosen as small as possible, while taking into account that the resource holding time may increase the subsystem utilization for small subsystem periods, and the overhead of context-switch will increase when selecting a smaller subsystem period.

In addition, we have extended the analysis of HSRP allowing subsystems to be developed independently. Also, we have proposed a new version of the overrun mechanism that in certain cases performs better than the other two overrun mechanisms proposed by HSRP. We have compared the three versions of overrun mechanisms based on their schedulability analysis and we have shown which parameters have greater effect on the system overhead. For example, to decrease this overhead, I) the resource holding time should be as small as possible, II) the subsystem period should be much greater than the resource holding time, and III) the subsystem period should be less than the smallest internal task period.

Furthermore, we have evaluated and compared the performance of SIRAP...
Chapter 4

Summary, Conclusions and Future Work

In this thesis we have addressed the problem of supporting global shared resources in the context of hierarchical scheduling. For this purpose, we have presented a novel synchronization protocol called Subsystem Integration and Resource Allocation Policy (SIRAP), which provides temporal isolation between subsystems that share logical resources. We have evaluated the overhead introduced by SIRAP through a simulation study. To decrease this overhead, the results of the study showed that the subsystem period should be chosen as small as possible, while taking into account that the resource holding time may increase the subsystem utilization for small subsystem periods, and the overhead of context-switch will increase when selecting a smaller subsystem period.

In addition, we have extended the analysis of HSRP allowing subsystems to be developed independently. Also, we have proposed a new version of the overrun mechanism that in certain cases performs better than the other two overrun mechanisms proposed by HSRP. We have compared the three versions of overrun mechanisms based on their schedulability analysis and we have shown which parameters have greater effect on the system overhead. For example, to decrease this overhead, I) the resource holding time should be as small as possible, II) the subsystem period should be much greater than the resource holding time, and III) the subsystem period should be less than the smallest internal task period.

Furthermore, we have evaluated and compared the performance of SIRAP
and *Overrun* by means of simulation analysis. The results of the simulation studies showed that in general, it is not trivial to evaluate which protocol is better than the other and the performance is extremely dependent on the subsystem and task parameters. In general, our evaluation shows that the *Skipping* mechanism used by SRIAP can perform better than the two mechanisms of *Overrun* if the task periods are much larger than their corresponding subsystem period. On the other hand, for a high difference between subsystem periods, the version of *Overrun* without payback can give better results. For equal or close to equal subsystem periods, the version of *Overrun* without payback performs better.

Based on the evaluation results, we could identify some sources of pessimism in the analysis of SIRAP and HSRP, and we have proposed tighter and more complex analysis. For SIRAP, we have proposed two different tighter analyses; one based on adding the effect of using SIRAP on the request bound function, and the other is based on adding the effect of using SIRAP on the supply bound function. The simulation studies showed that both proposed analyses can significantly decrease the CPU resource requirement when the number of accesses to a shared resource of a subsystem is high.

For *Overrun* (without payback), we have presented a tighter analysis, more accurately considering the limited preemptions from higher priority subsystems during overrun. In spite of the higher complexity of the new analysis, it can achieve a significant improvement in the CPU resource usage in some cases, especially when the ratio between resource holding time and subsystem period is high, which makes the performance of the traditional *Overrun* without payback very low.

Finally, we have studied the relationship between certain subsystem parameters that have great effect on the performance of the protocols, and we have proposed algorithms and design approaches that manipulate the subsystem parameters in order to improve the performance of the protocols in terms of requiring less CPU resource.

For SIRAP, we have presented an algorithm that finds the best internal ceilings of the global shared resources during the self-blocking state of a subsystem, and thereby minimizing the CPU resource requirement. We have shown through a simulation study that using the algorithm can decrease the overhead of using SIRAP in terms of requiring less CPU resource, specially for subsystems that have high subsystem periods.

For *Overrun* (without payback), we have identified a tradeoff between reducing resource holding time and increasing subsystem budget and its effect on the system load. We have showed that it is not possible to use this trade-
off during the development phase of a subsystem to minimize the system load, without providing the timing parameters of all other subsystems of the system. To effectively explore such tradeoff, we have presented a two step approach, where in the first step, we have proposed an optimal algorithm to generate a set of timing parameters (interfaces) during the development phase of a subsystem. In the second step and during the integration phase of subsystems, we have proposed an algorithm that can select the best interface from each subsystem to minimize the system load of the system.

4.1 Conclusions

To conclude, supporting global resource sharing is complex and may significantly increase the overhead when using a synchronization protocol. For instance, when using a synchronization protocol, and if the parameters of the subsystem are not selected carefully, it is not possible to guarantee schedulability of a subsystem even if its corresponding task set utilization is low. It is obvious that decreasing certain parameters can decrease the extra overhead of using synchronization protocols, such as decreasing the subsystem period, subsystem budget and resource holding times. However, unfortunately, the relationship between these parameters are so complex that decreasing one parameter may increase another parameter. Moreover selecting the optimal parameter depends on other subsystems. In this thesis we tried to provide some guidelines and methods for the designers of the subsystems to select the best synchronization protocol and the best subsystem parameters to decrease the overhead as much as possible.

4.1.1 Discussion

In the following text, we will discuss the applicability of each single contribution that we have presented, and also the possibility of generalizing and combining them.

Starting from the two considered protocols, SIRAP and Overrun, SIRAP handles the problem of budget depletion (explained in section 3.2) within the subsystem level. For Overrun, the budget depletion problem is considered in the global level by overrunning the given budget of a subsystem. This difference has many consequences on the analysis and implementation of each protocols. For instance, in SIRAP any scheduler in the global level providing SRP (if the global scheduler is of preemptive type) can be used without modifi-
cation. In addition, no communication between the local and global schedulers is required which provides better isolation between local and global schedulers. For Overrun, the global scheduler should be informed by the local scheduler when an overrun occurs, and this requires extending the implementation of the global scheduler. Since the effect of using SIRAP is handled locally within the subsystem level, its overhead in terms of requiring extra CPU resources, is added into the local schedulability analysis. This makes it possible for SIRAP to consider the behavior of tasks including the frequency of accessing global shared resources and the critical section execution time, in the local analysis to optimize the local schedulability analysis. This possibility has been used in the tighter analysis presented in paper D. For Overrun, and since its overhead should be considered in the global schedulability analysis, the task information should be included in the subsystem timing interface to be able to optimize the global scheduler considering the frequency of accessing global shared resources and the critical section execution time.

To decrease the overhead introduced by the proposed synchronization protocols, we have presented tighter analyses and algorithms that may be used during runtime and/or off-line depending on the type of the system. In general, systems can be classified as static and dynamic systems. Static systems are developed off-line and they do not change during runtime, while for dynamic systems, subsystems and/or tasks may be added or removed during runtime. We refer to the systems that their subsystems can be added or removed as Subsystem Level Dynamic (SLD). Usually, an admission controller is used to verify the possibility of adding a new subsystem without violating the schedulability of the subsystems. The admission controller applies the compositional analysis (global schedulability analysis) whenever a new subsystem is added. If tasks are allowed to be added or removed from a subsystem during runtime, then the systems is called Task Level Dynamic (TLD). Whenever a task is added to a subsystem, an admission controller re-computes the timing interface of the subsystem, and then it applies the compositional analysis.

For static systems all analysis is done off-line. Hence, the presented tighter analyses and algorithms for both SIRAP and Overrun (without payback) are applied during the design time of subsystems. For SLD, the subsystem analysis is performed off-line while the compositional analysis is done during runtime. The new analysis and the algorithm presented for SIRAP (in paper D and paper F) are applied at the subsystem level, i.e., they can be applied off-line during the design of the subsystem. On the other hand, the tighter analysis and the algorithm (presented in paper E and paper G) for Overrun (without payback) should be done at the system level, i.e., they have to be performed
online. Note that the presented analyses and the algorithms give better result than the original analysis, however, they increase the computation complexity of the analysis during runtime. To decrease the computation complexity during runtime, the admission controller may first apply a simple and less accurate analysis for faster response, and if the system is not composable it may use the more advanced analysis and algorithms. Considering TLD systems, all proposed analysis and algorithms should be applied during runtime.

Furthermore, let us discuss the possibility of combining the tighter analysis and the proposed algorithm for each proposed protocol. For Overrun, combining the tighter analysis and the algorithms of the two step approach significantly increases the computation complexity since the computation complexity of evaluating the system load when considering the tighter analysis is relatively high. For SIRAP, combining the presented algorithm and the new analysis may not increase the computation complexity since they do not affect each other.

Finally, considering the possibility of generalizing the two step approach (presented in Paper G) which was proposed to explore the tradeoff between decreasing resource holding time and subsystem budget, using Overrun (without payback). A similar tradeoff can be founded in SIRAP, however, because of high dependencies between some parameters used in the analysis of SIRAP (subsystem budget and resource holding time), the algorithms may not be able to find optimal settings. Nevertheless, we could prove in [35] that the two proposed algorithms can be used without modification for BROE as well.

4.2 Future work

The work presented in this thesis has left and opened some issues that would be interesting to investigate in the future.

- **Multi-processor**: The work presented in this thesis is suitable only for systems executed on a single processor while multi-processor architectures are becoming more attractive for real-time applications. Recently, there has been some focus on extending the hierarchical scheduling approach to multi-processor platforms [36, 37]. However, the problem of sharing logical resources has not been considered. It would be very interesting to generalize this work by including support for sharing of logical resources. As an initial step in this area, we have proposed a synchronization protocol for hierarchically scheduled multi-core systems in [38]. The presented protocol groups dependent tasks that directly or
indirectly share mutually exclusive resources into independent components. Within a component, dependent tasks use classical uni-processor synchronization protocols, such as SRP. The components are then scheduled on the cores by a global scheduler.

- **Multi-level HSF**: Another interesting work will be investigating the efficiency of using the synchronization protocols in multi-level hierarchical scheduling frameworks, since we only consider a two-level hierarchical scheduling framework. In [39] we present a schedulability analysis algorithm for FTT Ethernet enabled switches. The framework is a multilevel hierarchical scheduling framework and it uses the skipping mechanism to avoid pre-emption during message transmission which makes the analysis similar to the analysis of SIRAP.

- **Resources**: In this thesis, we have only focused on optimizing the CPU resource usage. However, considering other types of resources can be interesting and important, e.g., network, memory, power consumption, etc.

- **Approximation algorithms**: For task level dynamic systems TLD, the subsystem interface parameters should be recalculated during runtime, which may require faster algorithms. An efficient approximation algorithm has been proposed for the HSF [23] which significantly decreases the computation time of the local schedulability test, assuming that tasks are independent. A similar algorithm has been extended to be used with BROE in [40] and a simulation study showed that the pessimism generated from the approximation is very low. It would be interesting to generalize the use of this algorithm with SIRAP and HSRP.

- **Implementation**: Most of the recent implementations of the HSF presented in [32, 33, 41] do not separate the execution of the global scheduler and the local schedulers, and all these schedulers should be supported by the operating system. The motivation behind this way of implementation is that it is easier and more efficient to implement both schedulers together. One of the drawbacks of such an implementation is that the subsystems become platform dependent and if a subsystem uses a special scheduler, then the platform should support it. It would be interesting to implement a platform independent local schedulers within their subsystems and compare this approach with our existing implementations.
• **BROE**: We have compared the performance of SIRAP and *Overrun* by means of simulation analysis. We are planning to extend the comparison to include the BROE server. As a first step, we have adapted the analysis of BROE in [35, 40] to be compatible with our framework and the assumptions that we have considered when we compared between SIRAP and *Overrun*.

• **Subsystem period**: Finding optimal values for $P_s$ and $Q_s$ that minimize the overall processor requirement of the system, without providing information about other subsystems, is very complex and it requires extensive search algorithms [22] even for a hierarchical scheduling framework without resource sharing. Adding the problem of logical resource sharing makes it even more complex. It is an interesting research direction that could be investigated.
Chapter 5

Overview of the Papers

5.1 Paper A

Moris Behnam, Insik Shin, Thomas Nolte, Mikael Sjödin,
SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems,

Summary

This paper presents a protocol for resource sharing in a hierarchical real-time scheduling framework. Targeting real-time open systems, the protocol and the scheduling framework significantly reduce the efforts and errors associated with integrating multiple semi-independent subsystems on a single processor. Thus, our proposed techniques facilitate modern software development processes, where subsystems are developed by independent teams (or subcontractors) and at a later stage integrated into a single product. Using our solution, a subsystem need not know, and is not dependent on, the timing behaviour of other subsystems; even though they share mutually exclusive resources. In this paper we also prove the correctness of our approach and evaluate its efficiency.

Contribution

The basic idea of this paper was suggested by Moris Behnam. The work was mainly done in cooperation with Moris and Insik Shin, and Moris was responsible for the evaluation part of the paper and he was also involved in the schedulability analysis. All authors contributed to the writing.
Chapter 5

Overview of the Papers

5.1 Paper A


Summary  This paper presents a protocol for resource sharing in a hierarchical real-time scheduling framework. Targeting real-time open systems, the protocol and the scheduling framework significantly reduce the efforts and errors associated with integrating multiple semi-independent subsystems on a single processor. Thus, our proposed techniques facilitate modern software development processes, where subsystems are developed by independent teams (or subcontractors) and at a later stage integrated into a single product. Using our solution, a subsystem need not know, and is not dependent on, the timing behaviour of other subsystems; even though they share mutually exclusive resources. In this paper we also prove the correctness of our approach and evaluate its efficiency.

Contribution  The basic idea of this paper was suggested by Moris Behnam. The work was mainly done in cooperation with Moris and Insik Shin, and Moris was responsible for the evaluation part of the paper and he was also involved in the schedulability analysis. All authors contributed to the writing
5.2 Paper B


**Summary** The Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework to enable compositional schedulability analysis of embedded software systems with real-time properties. In this paper a software system consists of a number of semi-independent components called subsystems. Subsystems are developed independently and later integrated to form a system. To support this design process, in the paper, the proposed methods allow non-intrusive configuration and tuning of subsystem timing-behaviour via subsystem interfaces for selecting scheduling parameters. This paper considers three methods to handle overruns due to resource sharing between subsystems in the HSF. For each one of these three overrun methods corresponding scheduling algorithms and associated schedulability analysis are presented together with analysis that shows under what circumstances one or the other is preferred. The analysis is generalized to allow for both Fixed Priority Scheduling (FPS) and Earliest Deadline First (EDF) scheduling. Also, a further contribution of the paper is the technique of calculating resource-holding times within the framework under different scheduling algorithms; the resource holding times being an important parameter in the global schedulability analysis.

**Contribution** The paper is based on an idea of Insik Shin but Moris has done most of the work including the schedulability analysis for enhanced overrun mechanism and the comparison between the enhanced and the basic overrun mechanism, as well as the simplified equation to evaluate the resource holding times with the required proofs. All authors contributed to the writing of the paper.
5.3 Paper C


**Summary** Recently, two SRP-based synchronization protocols for hierarchically scheduled real-time systems based on Fixed-Priority Preemptive Scheduling (FPPS) have been presented, i.e., HSRP [9] and SIRAP [4]. Preventing depletion of budget during global resource access, the former implements an overrun mechanism, while the latter exploits a skipping mechanism. A theoretical comparison of the performance of these mechanisms revealed that none of them was superior to the other, as their performance is heavily dependent on the systems parameters. To better understand the relative strengths and weaknesses of these mechanisms, this paper presents a comparative evaluation of the depletion prevention mechanisms overrun (with or without payback) and skipping. These mechanisms are investigated in detail and the corresponding system load imposed by these mechanisms is explored in a simulation study. The mechanisms are evaluated assuming FPPS and a periodic resource model [23]. The periodic resource model is selected as it supports locality of schedulability analysis, allowing for a truthful comparison of the mechanisms. Given system characteristics, guiding the design of hierarchically scheduled real-time systems, the results of this paper indicate when one mechanism is better than the other and how a system should be configured in order to operate efficiently.

**Contribution** The paper was based on ideas of Moris and Reinder and Thomas. Moris has done most of the work including simulations and the analysis of the results. All authors contributed to the writing of the paper.

5.4 Paper D


**Summary** In this paper we have developed a new schedulability analysis for hierarchically scheduled real-time systems executing on a single processor
using SIRAP; a synchronization protocol for inter subsystem task synchronization. We have shown that it is possible to bound the number of selfblocking occurrences that should be taken into consideration in the schedulability analysis of subsystems, and correspondingly developed and proved correctness of two novel schedulability analysis approaches for SIRAP. An evaluation suggests that this new schedulability analysis can decrease the analytical subsystem utilization significantly.

Contribution The paper was based on ideas of Moris and Reinder. Moris has done most of the work including the schedulability analysis and the evaluation. All authors have contributed to the writing of the paper.

5.5 Paper E


Summary In this paper, we show that both global as well as local schedulability analysis of synchronization protocols based on the stack resource protocol (SRP) and overrun without payback for hierarchical scheduling frameworks based on fixed-priority pre-emptive scheduling (FPPS) are pessimistic. We present improved global and local schedulability analysis, illustrate the improvements by means of examples, and show that the improved global analysis is both uniform and sustainable. We evaluate the improved global and local schedulability analysis based on an extensive simulation study and compare the results with the existing analysis.

Contribution The paper is based on an idea of Reinder. The analysis was developed by Reinder but Moris was responsible for evaluating the new analysis and finding the parameters that affect the improvement of the new analysis. All authors have contributed to the writing of the paper.
5.6 Paper F


Summary In recent years, several synchronization protocols for resource sharing have been presented for use in a Hierarchical Scheduling Framework (HSF). An initial comparative assessment of existing protocols revealed that none of the protocols is superior to the others and that the performance of a protocol heavily depends on system parameters. In this paper, we aim at efficiency improvements of the synchronization protocol SIRAP and its associated schedulability analysis, where efficiency refers to calculated CPU resource needs. The contribution of the paper is threefold. Firstly, we present an improvement of the schedulability analysis for SIRAP, which makes SIRAP more efficient. Secondly, we generalize SIRAP by distinguishing separate resource ceilings for self-blocking and resource access. Using a separate resource ceiling for self-blocking enables a reduction of the interference from lower priority tasks, which can result in efficiency improvements. The efficiency improvement depends on both subsystem characteristics and the value selected for the resource ceiling for self-blocking, however. The third contribution of this paper is therefore an algorithm that given a subsystem selects for each globally shared resource an optimal value in terms of efficiency for its resource ceiling for self-blocking. The efficiency improvement gained by the algorithm compared to the original SIRAP approach is evaluated by means of simulation.

Contribution The paper was based on ideas of Moris and Reinder. Moris was responsible for developing the algorithm and evaluating its performance. All authors have contributed to the writing of the paper.

5.7 Paper G

Summary  This paper presents algorithms that (1) facilitate system independent synthesis of timing-interfaces for subsystems and (2) system-level selection of interfaces to minimize CPU load. The results presented are developed for hierarchical fixed-priority scheduling of subsystems that may share logical recourses (i.e., semaphores). We show that the use of shared resources results in a tradeoff problem, where resource locking times can be traded for CPU allocation, complicating the problem of finding the optimal interface configuration subject to schedulability. This paper presents a methodology where such a tradeoff can be effectively explored. It first synthesizes a bounded set of interface-candidates for each subsystem, independently of the final system, such that the set contains the interface that minimizes system load for any given system. Then, integrating subsystems into a system, it finds the optimal selection of interfaces. Our algorithms have linear complexity to the number of tasks involved. Thus, our approach is highly suitable for adaptable and reconfigurable systems.

Contribution  The paper was based on ideas of Moris and Insik. Moris was responsible for developing the algorithms and proving their correctness and optimality formally. Moris was also involved in the discussions and witting of the other parts of the paper. All authors have contributed to the writing of the paper.
Chapter 5. Overview of the Papers

Summary
This paper presents algorithms that (1) facilitate system independent synthesis of timing-interfaces for subsystems and (2) system-level selection of interfaces to minimize CPU load. The results presented are developed for hierarchical fixed-priority scheduling of subsystems that may share logical resources (i.e., semaphores). We show that the use of shared resources results in a tradeoff problem, where resource locking times can be traded for CPU allocation, complicating the problem of finding the optimal interface configuration subject to schedulability. This paper presents a methodology where such a tradeoff can be effectively explored. It first synthesizes a bounded set of interface-candidates for each subsystem, independently of the final system, such that the set contains the interface that minimizes system load for any given system. Then, integrating subsystems into a system, it finds the optimal selection of interfaces. Our algorithms have linear complexity to the number of tasks involved. Thus, our approach is highly suitable for adaptable and reconﬁgurable systems.

Contribution
The paper was based on ideas of Moris and Insik. Moris was responsible for developing the algorithms and proving their correctness and optimality formally. Moris was also involved in the discussions and writing of the other parts of the paper. All authors have contributed to the writing of the paper.

Bibliography


