Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

---

Thesis for the Degree of Master of Science in Computer Science with Specialization in Embedded Systems 30.0 credits

# OFFLINE SCHEDULING OF TASK SETS WITH COMPLEX END-TO-END DELAY CONSTRAINTS

Jonas Holmberg
jhg12006@student.mdh.se

Examiner: Saad Mubeen
Mälardalen University, Västerås, Sweden

Supervisor: Matthias Becker
Mälardalen University, Västerås, Sweden

June 9, 2017

## Abstract

*Software systems in the automotive domain are generally safety critical and subject to strict timing requirements. Systems of this character are often constructed utilizing periodically executed tasks, that have a hard deadline. In addition, these systems may have additional deadlines that can be specified on cause-effect chains, or simply task chains. They are defined by existing tasks in the system, hence the chains are not stand alone additions to the system. Each chain provide an end-to-end timing constraint targeting the propagation of data through the chain of tasks. These constraints specify the additional timing requirements that need to be fulfilled, when searching for a valid schedule. In this thesis, an offline non-preemptive scheduling method is presented, designed for single core systems. The scheduling problem is defined and formulated utilizing Constraint Programming. In addition, to ensure that end-to-end timing requirements are met, job-level dependencies are considered during the schedule generation. Utilizing this approach can guarantee that individual task periods along with end-to-end timing requirements are always met, if a schedule exists. The results show a good increase in schedulability ratio when utilizing job-level dependencies compared to the case where job-level dependencies are not specified. When the system utilization increases this improvement is even greater. Depending on the system size and complexity the improvement can vary, but in many cases it is more than double. The scheduling generation is also performed within a reasonable time frame. This would be a good benefit during the development process of a system, since it allows fast verification when changes are made to the system. Further, the thesis provide an overview of the entire process, starting from a system model and ending at a fully functional schedule executing on a hardware platform.*

# Table of Contents

# List of Figures

## Glossary

| Notation | Description |
|---|---|
| End-to-End Deadline | Maximum allowed latency for a given chain. |
| End-to-End Latency | Total time for data to propagate through an entire chain. |
| Hyperperiod | Least common multiple of the periods for a set of tasks. |
| Job-Level Dependency | A ordering requirement defined over two different task instances. |
| Schedulability | Success rate obtaining a valid schedule for a given system. |
| Scheduling | The act of ordering the execution of tasks in a system. |
| Task Instance/Job | When defining a system model each task can be executed several times in, these parts are called instances or jobs. |
| Task-Chain | Also called cause-effect chain, dictates propagation of data in order to the tasks that make up the chain. |
| Task-Chain Congestion | A value describing the relationship between the number of tasks and chains in a given system. Accounting for the amount of tasks belonging to any chain along with the amount of overlap between several chains. |

## Acronyms

| Notation | Description |
|---|---|
| CP | Constraint Programming. |
| CSP | Constraint Satisfaction Problem. |
| HP | Hyperperiod. |
| IDE | Integrated Development Environment. |
| LCM | Least Common Multiple. |
| RTOS | Real Time Operating System. |
| WCET | Worst Case Execution Time. |

# 1 Introduction

Software design in the automotive domain is mostly a highly complex process, consisting of many demands that need fulfilment When dealing with real-time systems usually strict timing requirements exist on all tasks that make up any given system. Each task is divided into different instances, or jobs, during the execution. Thus, each task instance needs to fully execute within a certain timing window to ensure predictable behaviour with respect to deadlines. Additionally, systems of this variety are generally safety critical. System failures may result in serious damage, both property wise and personally. Therefore, during the development phase for systems of this nature it is of utmost importance that assurance can be made to the system integrity. Both in terms of operational validity for all tasks, along with ensuring that execution is conducted within any set time period.

Systems of this character are often designed using periodically occurring tasks, where each of the tasks has its own respective deadline. Additionally, these types of systems can also be subject to end-to-end timing constraints. Meaning that timing constraints also exist on chains of independent tasks, in addition to their own internal timing constraints. Tasks involved in such chains could be assigned different periods, resulting in over and under-sampling situations. In an over-sampling situation the preceding task for any task in a given chain would operate at a higher period than the succeeding task. Resulting in data being overwritten and not being utilized. Further, an under-sampling situation behaves in an opposite manner. Where for any task in a chain the preceding task operates at a lower period. The consequence of this is that some data is utilized for several task executions. Thus, in turn making analysis of the system non trivial [1] [2].

Commonly in industry today, the design process moves from the system model to either simulation of the complete system or deployment on an actual platform. When this stage has been reached it is possible to investigate all the end-to-end delays present in the given systems. For both cases this can prove to be quite expensive. Considering in this case, regardless of scheduling approach, the schedule would need to be executed to fully evaluate the schedulability of the given system model. In [3] an approach how to analyse such systems is presented, without any prior knowledge of the underlying hardware or the scheduling algorithm used. Further, the paper presents how to generate so called job-level dependencies, representing a partial ordering of task instances within one hyperperiod. Their objective is to remove certain scenarios where the end-to-end delay would be longer than the specified timing requirements. Accounting for these job-level dependencies when scheduling a system will ensure end-to-end timing requirements are met. In addition, this provides the possibility to perform analysis on end-to-end delay during the development process. This means that the software model itself can be analysed. Moreover, since this can be done at earlier stages during development it carries the potential to reduce the cost. These concepts are implemented in the MECHAniSer tool presented in [4]. The concept of job-level dependencies provides the starting point for the thesis.

This thesis presents an approach to implement a solution how offline scheduling can be applied for system models with specific end-to-end timing requirements. To ensure the given timing requirements are met, the concept of job-level dependencies will be utilized during the schedule generation. Currently there is no method in place for offline scheduling generation, that takes the job-level dependencies presented in [4] into account. The thesis aims to present an approach that can produce offline schedules, ensuring end-to-end timing requirements are met by taking job-level dependencies into account during the generation process. This solution would provide an efficient approach to generate schedules that have no need for real execution to ensure all timing requirements. Considering any valid schedule generated utilizing the job-level dependencies will ensure all end-to-end timing requirements. As mentioned, job-level dependencies do not account for any underlying hardware or method used for scheduling. Hence, one significance of this work is that an efficient way of testing if any given system model will be schedulable for a particular hardware platform is presented. Further, the presented approach can also verify schedulability without necessary deployment to hardware or simulation. Because the schedule is generated offline everything is already defined and it will be known if all timing requirements are met. Nevertheless, the work will also present how a generated schedule can be deployed to hardware, showing that the entire process works.

## 1.1 Problem Formulation

The overarching problem that presents itself in this work is how to approach the scheduling method. The system models to be scheduled in this work have, in addition to timing requirements on individual tasks, extra timing requirements on chains of tasks. On each chain a so called end-to-end timing constraint, that also needs to be satisfied for a schedule to be considered valid, is specified. Fulfilment of these additional conditions is an important part of the main problem.

As this thesis focuses on applying an offline scheduling approach to the type of system models presented, the main problem is how this should be performed. Further, the paradigm of Constraint Programming (CP) will be used to define the scheduling problem. Therefore that problem is extended to how CP can be applied to define the schedules representative of the given system models. Moreover, ensuring that the generated schedules meet all timing requirements, both for individual tasks along with any end-to-end timing constraints, is of importance. Also, how job-level dependencies can be translated and applied to the CP solution, is a part of the problem.

Furthermore, the generated schedule needs to be executed on real hardware for verification and testing purposes. Thus, it becomes necessary to implement an execution framework on a suitable hardware platform. Afterwards the schedule can be evaluated to see if it performs as anticipated, meaning all end-to-end constraints are met. Several leading aspects to observe include the time it takes to generate a suitable schedule. Further, it will be of interest to analyse how this time changes in regards to task set sizes, the number of job-level dependencies to name a few. This is interesting to examine since CP does face a scalability problem, that could render larger problem sizes unmanageable. Although CP in combination with timed intervals require less decision variables and therefore should scale better against larger problem sizes.

## 1.2 Outcomes and Goals

The outcomes of this thesis are divided into different parts. This includes both technical and theoretical outputs. Thus, they can be described as follows:

- A defined approach to generate a scheduling problem for any given system model. The ability to translate a system model along with job-level dependencies into the CP domain, enabling solving for a valid schedule.
- The ability to interoperate the result provided from the solved CP problem into a finalized schedule. Including generation of an output file for any given schedule that can be attempted to be executed on real hardware.
- An extended operating system that can run on real hardware with the ability to execute the offline generated schedule.
- An extensive evaluation of the generated schedules. Most of the evaluation performed on more advanced system models, along with some easier models for hardware testing. Evaluation includes, examine the behaviour of the end-to-end delays as a result of the job-level dependencies. Compare to sets without dependencies to observe if actual improvement can be detected. Determine why the results turn out a certain way. Also perform some examination on the schedule generation times.

## 1.3 Research Questions

Considering no current proposed method exist, regarding generation of offline schedules for system models utilizing job-level dependencies to meet end-to-end delay, this is the main area of contribution presented in this thesis. Hence, to further assess what this contribution entails, research questions are formulated that aim to provide answers on the robustness and efficiency of the proposed approach. The overarching questions are defined as follows:

1. Is it possible to develop a method that can efficiently generate valid schedules using CP, based on a given system model containing the number of tasks and chains, along with associated job-level dependencies?

   (a) How efficient will CP be in the given scenario in terms of generating valid schedules? Considering both success rate along with performance.
   (b) How well the schedule generation approach scales in regards to the size of a given system model? Further how will the number of needed constraints scale in regards to the system models?
   (c) Will there be an observable difference in the end-to-end delay when utilizing job-level dependencies, as opposed to when not?

2. How will the generated schedule actually behave in a real environment? Will it execute within the confounds that were expected from the pre generation?

Majority of the focus in this thesis will be aimed towards the efficiency of the scheduling approach. Evaluation of its success rate and performance will be presented as the main result. Secondary to this is the evaluation of the approach on real hardware. Considering this is not the main focus of the thesis, these results will be presented to provide a proof of concept that the entire chain: going from model to finished executing schedule on real hardware works.

## 1.4 Report Outline

Firstly in Section 2 the report will provide the necessary background in the area and provide information on some of the main concepts this thesis encapsulates. Further, in Section 3 related work done in this area will try to be assessed to provide some understanding of what has been done and what are some of the proposed approaches to tackle the types of problems envisioned in this thesis. Subsequently the method utilized for the thesis will be discussed in Section 4. The technical approach will be presented in Section 5 and 6, aiming to provide additional detail how the scheduling was performed. Afterwards, the conducted testing and evaluation will be presented in Section 7, along with a discussion part reasoning about the results in Section 8. Lastly, the final conclusion will be presented along with potential future work, in sections 9 and 10 respectively.

# 2 Background

Ensuring the runtime requirements of a real-time system demands that the associated tasks get scheduled in an appropriate manner. Depending on the type of system, the requirements, restrictions and demands it presents can affect what type of scheduling approach is most suitable. Choosing the appropriate solution for the problem can be vital, since the demands on these systems can be harsh in regards to task execution times and safety critical aspects. Depending on the usage area of systems in this category, failure to ensure the distinct requirements that are demanded, can result in unfortunate situations. Therefore, regardless of the selected scheduling approach, assurance in regards to performance needs to be provided. Particularly concerning timing requirements defined by the tasks present in any given system. Further, when considering the type of systems covered in this work, it is also important to meet end-to-end timing requirements.

The scheduling problem tackled in this thesis is mostly based on definitions and requirements defined by the automotive industry. Therefore, comparisons and research made in this work is mostly aimed towards automotive solutions and implementations. Generally this should relate more to what this work entails compared to other domains. Nevertheless, other areas are not necessarily omitted because of this. Considering, real-time systems and approaches to scheduling in this domain and the requirements are often quite similar overall.

In this thesis the scheduling approach was predetermined in regards that it were to be solved with an offline approach. What this entails is the manner how the schedule is obtained. Commonly scheduling is divided into two different categories, firstly there is online scheduling. For this approach the decisions in regards to what the system should be doing at any given moment is decided at run-time. In contrast to offline scheduling, also commonly referred to static or deterministic scheduling, the ordering of operations in a system have been defined ahead of time. Performing scheduling offline requires that essentially all characteristics of the target system is known beforehand. Otherwise, generating a schedule that will be able to ensure all timing requirements will be very difficult. Considering this thesis presents an offline solution, the type of system targeted is predefined. The schedules produced are intended to execute on a single core system. Additionally, all periods, execution times and deadlines are also constant and non changing over the course of system execution. Another important restriction to consider is that preemption is not supported. Had these aspects not been predefined or known, defining a schedule offline would not have been accomplishable.

## 2.1 Constraint Programming

Constraint programming is an approach to programming where relationship, between various parts of the system and its variables, is defined by various constraints. Hence, constraints restrict and limit the available values that a variable can take. Essentially the constraints provide some partial information about the variables it governs over. Therefore, the main idea by this approach is to solve problems by defining a set conditions representing the given problem, that subsequently needs to be fulfilled for a valid solution to be obtained. The constraints together form a Constraint Satisfaction Problem (CSP) that states what relations needs to hold among the variables [5]. A CSP is the first step in solving a problem with the help of CP. This part is referred to as modelling [6]. In relation to scheduling, this would entail translating the system model into the CP domain. When defined, the problem can be solved for in attempt to locate a solution that fulfils the specified CSP.

Aiding in the solving of problems defined with the help of CP, several different solvers exist that can make this process more streamlined. One of the more prominent examples of such a solver is the IBM ILOG CP Optimizer [7], that is utilized in this thesis. These solvers will take problems formulated with constraints and then analyse them for possible solutions as result of the defined conditions.

### 2.1.1 Conditional Time Intervals

Conditional Time Intervals [8] [9] allows for the creation of so called interval variables. Since this work considers usage for scheduling, the variables will be related to that given area of usage. Interval variables define an interval of time wherein it can be valid. Relating this to scheduling,

| Constraint | Description, Interval variable $A$ and $B$ |
|---|---|
| startBeforeStart | $A$ must start before $B$ allowed to start |
| startBeforeEnd | $A$ must start before $B$ allowed to end |
| endBeforeStart | $A$ must end before $B$ allowed to start |
| endBeforeEnd | $A$ must end before $B$ allowed to end |
| startAtStart | $A$ must start when $B$ starts |
| startAtEnd | $A$ must start when $B$ ends |
| endAtStart | $A$ must end when $B$ starts |
| endAtEnd | $A$ must en end when $B$ ends |

Table 1: Precedence constraints, that can be applied on interval variables.

the interval variable will represent a value in time where part of schedule can be executed. This includes the ability to define start and end values for the interval that both require a minimum and maximum value. Afterwards, a number of precedence constraints can be added to define the ordering of various interval variables. In Table 1 the precedence constraints can be seen. Using these constraints can dictate how the different interval variables are allowed to be ordered. These constraints will finalize the problem that needs to be solved. Hence, a system model that should be scheduled needs to be translated into a number of interval variables and constraints. Subsequently, when solving the defined constraint problem a valid schedule should be generated. Lastly, additional types of constraints exist, but for the work presented in this thesis they will not be necessary, therefore they will not be assessed. For further information refer to the associated papers [8] [9].

## 2.2 Real time system

For a real-time system arriving at the correct computational result is not the only important aspect, it is equally vital the result is obtained within at the correct time or within the specified deadlines. Another aspect that may often be the case for systems of this character is that they need to be safety critical. Meaning that they cannot fail under any circumstance. In some cases such errors could result in great damage physical or property wise. Ensuring that real-time systems fulfil these strict requirements they often need a handler for the jobs the system needs to perform, especially for larger systems, this is the assignment of the scheduler.

A real-time system can be either defined as a hard or soft system [10]. A hard real-time system dictates that the deadline cant be missed under any circumstance, otherwise it could cause some catastrophic failure or a total system failure. In contrast, a soft real-time system aim to meet all deadlines, although a miss would not cause any massive harm. Some additional characteristics can be viewed in chapter 1.5.1 in [10]. For this thesis, hard real-time system models are considered. Because the presented approach is angled towards the automotive industry it is a necessity, since in that domain systems are mostly of that characteristic. Hence, meeting all timing requirements is compulsory when generating schedules for systems with specification that falls within this domain.

Its not impossible for a real-time system to execute and perform its assignment without any form of scheduling approach. Although, in many cases when systems reach a certain size and its responsibilities grow the system cannot execute in the previous manner. At this point it should be considered to employ some real-time scheduling approach to assist the system in performing its

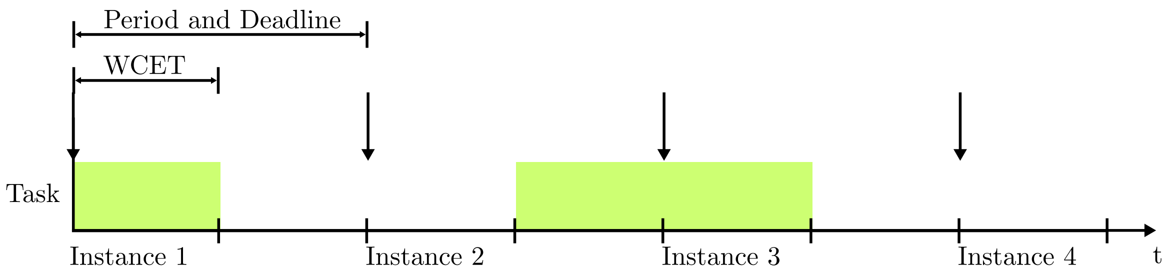


Figure 1: Showing an periodic task $\tau_i$ along with its associated period $T_i$, deadline $D_i$ and WCET $C_i$. In this example the deadline is the same as the period.

assignments correctly and ensure execution within possible timing requirements. For this to work, the assignments need to be divided into smaller parts. Therefore, the given system will be divided into a number of different tasks, that together should perform the intended purpose of the system. Tasks are generally defined with a period, deadline and a Worst Case Execution Time (WCET) wherein the task should finish its execution before the end of the given deadline. Ensuring that each task will confine within their given deadlines and that the right job is executed at the correct time is the true assignment of the scheduler. Performing scheduling is something that can be done using a variety of approaches. As mentioned earlier two main categories exist, online and offline, where a form of the latter is used in this thesis.

## 2.3 Tasks

Simply put, a task is responsible for a specific part of a system and performs some associated assignment. A system is generally composed of several different tasks that together represent all functionalities of the given system. Each one of these individual tasks in the system gets allocated runtime by the scheduler. A variety of different task types exist that differs somewhat in their definition and usage. Three common task types are:

**Periodic Tasks** are tasks that are triggered by time. As the name entails the activation of the task will occur periodically at a fixed time interval. Generally periodic tasks are very static in their behaviour, and thus behave in a predictable way. A system comprised of only periodical tasks is more predictable than a system that would utilize tasks triggered by events. Since it will always be known when a certain task should execute.

**Aperiodic Tasks** are tasks that have no defined period of execution. These tasks are commonly called event triggered tasks, meaning that execution can be initiated at any time. There is no limitation in regards to when tasks of this type can be activated. Further, such a task can be triggered for execution again but it cannot be known when and if it will be activated again. Hence, the behaviour is inherently unpredictable, since it cannot be known when the task gets ready to execute.

**Sporadic Tasks** , same as aperiodic, are time triggered and can be activated at any time. Between the two one major difference exist in regards to what occurs once an execution event has been finished. When an aperiodic task has finalized its execution it will be locked from further activation until a given periodic time has passed, referred to as minimum interarrival time. Afterwards the task will be eligible for activation again.

Considering the focus of this work is offline scheduling, only periodic tasks will be considered. The reason is that periodic tasks are more appropriate when dealing with offline scheduling, because of their predicable behaviour. A standard offline schedule needs to have all the execution defined before runtime, causing event triggered tasks to be non suitable. Solutions exist that can combine online and offline scheduling to utilize event triggered tasks [11] [12], but this work only consider pure offline scheduling. In Figure 1, a description of the type of tasks used in this work can be seen. A number of important terms, and how they apply to this work, that should be understood are defined as follows:

**Period** is the interval wherein any given task is allowed to perform its execution. Each period of a task defines when the so called release time should occur. This is the moment in time where the task is allowed to start its execution. This will happen continuously every time the amount of time defined by the period have passed, hence the name. Further, a task can never execute more than one time within a single period interval. Thus, he definition of the period will define the rate of execution of any task.

**Deadline** is a point in time where a given instance of a specific task needs to have completed its execution, in relation to the time that the task was released. In hard real-time systems, that this work falls under, it is vital that the deadlines are met. In this work, so called implicit deadlines are utilized. It means that a given deadline is the same as the period of a task, as can be seen in Figure 1.

**WCET** is the Worst Case Execution Time. It is the representation of the longest time any given task can spend executing. A given task may execute faster than this time, although the execution cannot be slower. When any given task starts to execute, the remaining time of the period needs to be longer than the WCET. If that is not the case, it is not possible to ensure that a task will always be able to meet its given deadline.

**Task Instance/Job** is one execution event of a single task in a system. In regards to offline scheduling the number of instances needed for each task in a system is decided based on the hyperperiod. The hyperperiod is determined based on the lowest common multiple of all tasks in any given system. Afterwards, the number of needed instances or jobs needed for each task can be calculated.

### 2.3.1 Task Chain

A cause-effect chain or simply a task chain, is defined by a number of tasks present in a given system. In the context of this work, the purpose of a task chain is to propagate data trough the entire chain. Further, in addition to a tasks individual deadline, any given task chain has its own defined deadline. Meaning, from the point in time where the head of the chain executes, the following tasks in the chain need to confine their own execution time within the chains defined timing constraint . Hence, an additional timing requirement needs to be fulfilled for a schedule to be considered valid. Task chains are at the very centre of this thesis since they are responsible for the so called end-to-end delay that represents the major scheduling problem this work tackles. Accounting for all the task chains and their deadlines, that can be present in a given system, is necessary in the schedule generation process.

## 2.4 End-to-End Delay

The term end-to-end delay can often be used in many different areas of work. A common usage of the term is when talking about networks, the time it takes for a package to be transmitted. Although, in this work it refers to the time data takes for tasks in a chain to execute. Essentially, what end-to-end delay means is the total time it takes from reaching point a to b. In the work performed in this thesis the delay is used to define the time it takes for data to propagate through chains of tasks as defined in [3]. An example of what this entails can be observed in Figure 2. In this example a chain of tasks consist of $\tau_1$, $\tau_2$ and $\tau_3$ where the data propagates in the same order as mentioned. The delay for the chain is defined by the time it takes the data to propagate through it entirely. In the example the total delay would end up at $t = 8$ time units. In this example it may appear trivial, although when more tasks are added along with an increase in the size of the chains it will turn out to be more complicated.

When the delay is calculated all time aspects that affects the chain needs to be taken into account. From the point the data becomes available from the first task in a chain, until it is written from the last. This defines how the End-to-End delay is obtained, simply from the start of the chain to the last point of the chain. This property is of prime importance in this work,

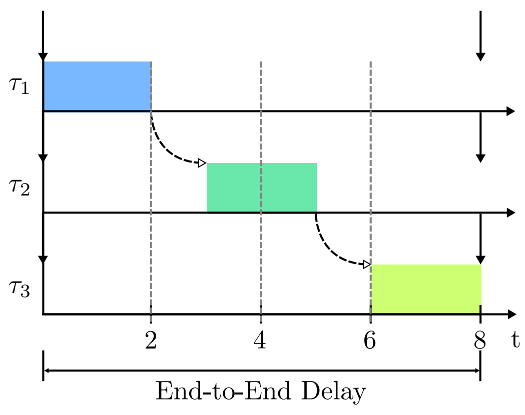


Figure 2: Showing what the End-to-End delay entails in regards to this work. $\tau_1$, $\tau_2$ and $\tau_3$ represents a task chain. The total End-to-End delay is the time it takes for the data to propagate through the entire chain.

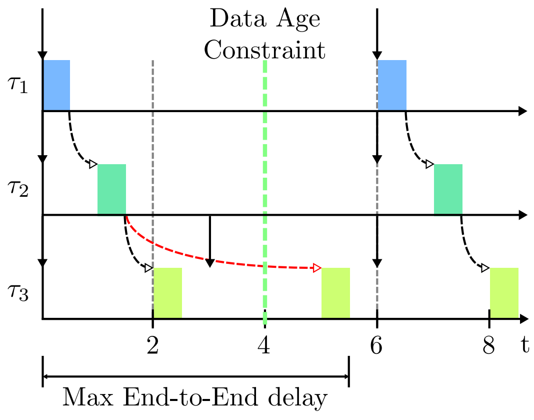


Figure 3: Showing where the maximum data age is overrun. When the second instance of $\tau_3$ reads data it reads data that is no longer valid. Hence, the End-to-End delay constraint is broken, and the schedule is not valid.

since much of the evaluation will be concluding observing the behaviour of this end-to-end delay for task chains in the scheduled system. If the end-to-end timing constraint cannot be met while scheduling a system it is not a valid schedule.

### 2.4.1 End-to-End Delay Constraint

Dealing with task chains that propagate data through the entire chain, a maximum allowed delay in a chain may be defined[13], creating additional timing requirements that need to be fulfilled. In other words, this is a limitation on the permitted age of valid data that can be used in a chain. Data age constraints specify the additional timing requirements that should ensure that the data used still can produce outcomes that are relevant to the current situation of the system and its environment. A simple example of this could be that one task in the system reads the current velocity of vehicle. Afterwards, this data propagates thorough a chain of interlinked task that utilize this information to form its own individual decisions on various parts of the system. For example one of these actions could be to calculate the distance travelled. If the age of the data would exceed a certain time limit it could be considered antiquated to accurately provide an estimation on the travelled distance.

Another example can be viewed in Figure 3, presenting a situation where the maximum allowed data age is exceeded. In the figure the maximum allowed data age is defined by the dashed green line, at a time value of $t = 4$. As can be seen the second instance of $\tau_3$ reads data that is older than the allowed, represented by the read arrow. Causing the total end-to-end delay to reach a time of 5.5 time units. This would be an invalid schedule since it does not meet the end-to-end timing requirement imposed by the end-to-end delay constraint.

Because of situations like these it is important to handle possible instances where the delay of a set of tasks exceed a proposed maximum timing constraint. When generating an offline schedule some definition of these constraints needs to be present if a suitable decision is to be made. In this work these delay constraints will be ensured when the scheduling is to be performed with the help of Job-Level Dependencies. Utilizing these dependencies in the generation of the schedule will help handle any provided End-to-End timing requirement that the constraints create on the system.

### 2.4.2 Job-Level Dependencies

Central to the scheduling of task sets in this work is the definition of job-level dependencies between tasks as defined in [3]. Utilizing these dependencies is of major importance to the work presented in this thesis, since it is one of the main conditions that needs to be satisfied when the schedule is being defined. The reason why the dependencies are used is because they provide a solution to aid in meeting all end-to-end timing requirements. Individual task deadlines in combination with these dependencies make up the major scheduling timing requirement that will be considered when constructing the offline schedule.

For this work the job-level dependencies used are aimed to target the maximum data age between different chains of tasks. This entails the propagation of data through a defined task chain. In many cases a maximum age of the data is defined because if the age of some data reaches a certain point it may not hold any information that is worth using any more. Because of this it is vital to make sure that the data will not be too old when it is allowed to be used. Considering this work tackles offline scheduling of task sets with these accompanying dependencies, it is even more important that some conditions help define these dependencies between the tasks. When performing the scheduling offline it is crucial that some condition can ensure that the end-to-end time requirement for the maximum data age between tasks is met. This is the purpose of the job-level dependencies. Hence, when defining the schedule also meeting the conditions of these dependencies along with individual task deadlines will ensure that the end-to-end delay is met. Observing the success rate of obtaining valid schedules meeting the end-to-end delay will be important to the evaluation. Also how these numbers change when different amount of tasks and chains are used in the system model.

The main purpose of the job-level dependences is to specify precedence constraints on the set of tasks in a way that will guarantee that the end-to-end delay constraints are met. Thus, restricting task ordering in a way that removes possible combinations that result in end-to-end delays that fall outside the set timing requirement. If job-level dependencies can be created for a given system it provides an indication that it is possible to meet the end-to-end requirements. Nevertheless, the dependencies on their own do not guarantee that a schedule can successfully be defined. Because certain conditions can be created that cannot be fulfilled together with the rest of the conditions that make up the entire system. Further, since the method described in [3] does not base any decisions in regards to the scheduling approach that may be used, the provided results are pessimistic. Meaning there could possibly exist combinations within the constraints that could produce a scheduling order with even lower delay. In this thesis that will be an interesting aspect to observe. How the generated schedules delay will differ compared to the worst case delays that will be ensured by the job-level dependencies. Further, because the underlying system or scheduling approach is not considered by the job-level dependencies, fulfilling them is not always possible. Hence, if job-level dependencies can be defined and their requirements can be fulfilled when creating the schedule, it will always meet the end-to-end requirements. But, the ability to define job-level dependencies does not equate that a valid schedule exist that can meet the demands of the dependencies. Understanding this aspect is important, otherwise one could assume that job-level dependencies on their own always equals a valid schedule, but that is not the case.

## 2.5 Offline scheduling

Considering this project aims to evaluate the use of offline scheduling as a schedulability approach to the mentioned complex task sets, some information on the area should be provided. With the intention and purpose of obtaining a good and valid approach how to generate these offline schedules in an efficient and accurate way, some existing methods and applications will be considered. Further, a general overview on the area of offline scheduling will be provided.

Offline scheduling entails the acquisition of a feasible execution order, for a given set of tasks, before the system has started operating. One major requirement in order for an offline schedule to be generated is the complete understanding of the system and the environment. Meaning that the system, number of tasks, chains, WCET and so forth needs to be known. Without this knowledge producing a feasible schedule is not possible, since no changes can be made at runtime. What an offline scheduling approach generates is essentially a table containing all the information regarding how the system should run. Also, since the execution order is generated offline, more complex task sets can be processed [14]. This ties in well to the requirements for this thesis. Considering the task sets to be scheduled include some complex constrains in regards to the end-to-end delay, offline scheduling is very suitable since certain conditions can be guaranteed.

Solutions generated by this approach are fully time triggered. Meaning that when executed, the system will simply select what task should run at each instance based on the timings provided by the generated execution table. This is handled by something called a dispatcher. The process

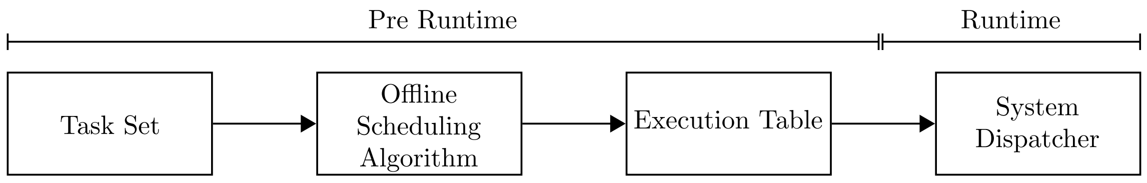


Figure 4: The process to get an offline scheduled task set to run.

can be viewed in Figure 4. Among the advantages of utilizing offline scheduling is the ability to make sure that the system will run as intended. This is beneficial for safety critical systems, since it can be assured that the schedule can meet all requirements before execution. Although one of the drawbacks is that if extensions are warranted to the system an entirely new schedule needs to be obtained. Depending on the scale of the additions, the time it could take to generate new functional schedules could be dramatically increased.

### 2.5.1 Approaches to Offline Scheduling

When defining an offline schedule the common approach is to find an ordering of the tasks in a set, within one hyperperiod. Afterwards, this generated order can be iterated continuously for the system to run within its defined bounds. When it comes to offline scheduling usually the entire system must be known beforehand, it needs to be predictable [15]. Without knowing the definitions of the system beforehand, creating any offline schedule that can be ensured to work is very challenging.

In [16] two approaches in regards to runtime representation of offline schedules is discussed. Additionally, this paper brings up a number of requirements that are necessary for the specification of an offline schedule. Runtime representation of the schedule is how the tasks in the system are organized to run and how the system should execute them on the platform at runtime. The first suggestion is to represent the tasks in the systems with vectors. Wherein one position represents a time where a task can initiate its execution. Further, the second approach to task runtime representation utilizes a matrix to represent the schedule. Each row represents a chain of tasks along with the instance wherein the dispatcher should start executing said chain. This reduces the amount of rescheduling that is bound to the system clock.

Finding suitable ordering of the tasks for the dispatch table can be very time consuming without an adequate approach. Because of this, a way is needed to efficiently locate a suitable schedule that can be represented by an execution table. Regarding a specific approach for how to obtain an offline schedule in this work had been decided before hand. The intended approach was to use constraint programming along with its time variables. Nevertheless, a number of different approaches are presented to provide a small overview of the possible methods that can be used to obtain a schedule.

One approach is to use a heuristic search to locate a suitable schedule. In [17] a dispatch table is divided up into several smaller parts called slots. The hyperperiod of the table needs to be evenly divisible by the slot size, so that a number of equally sized slots can be created. The paper uses an ”Least-loaded” heuristic algorithm to find suitable schedules. It entails dividing each task into what is defined as the least loaded slot. The current load of a slot is defined by the WCET of the tasks already placed inside a specific slot. The same paper also proposes another heuristic approach called ”Lowest Peak”.

One approach to offline scheduling is presented in [18]. Wherein complex constraints on a set of tasks are simplified to constraints suitable for earliest deadline first scheduling. Starttime and deadline make up the simple constraints. When these have been transformed they are scheduled with an EDF based bandwidth server. Only the first task in the chain are given a starting time, the following tasks in the chain will simply be executed in order after each other with no regard to any new start time.

# 3 Related work

Since there are a number of different areas that should be considered in this work the related work is divided into different sections. Firstly, to get a general overview of the area, standard approaches used in automotive software development will be assessed further. Following this, the aspect of offline scheduling will also be evaluated to provide a better understanding of that area, since offline scheduling is the main focus of the work performed in this thesis. Afterwards, Constraint Programming will also be in focus. It is relevant considering it is the chosen approach in regards to schedule generation. Hence giving a short overview of the usage can be useful.

## 3.1 Automotive software

Considering software design for automotive systems is a complex area with generally no room for anything to go wrong. These types of systems are highly time critical, meaning the majority of functions the system handles need to be processed with the correct timing in mind. The complexity is further increased in these types of systems because additional timing constraints can exist between a set of independent tasks. Because of this it is vital that the systems developed are thoroughly evaluated and tested, to assure that the strict requirements are sufficiently met. To provide an understanding of the different approaches that are used or could be used in regards to handle these time critical automotive systems, some major approaches will be discussed.

Currently in the automotive domain several different approaches to the scheduling of applications exist. When it comes to the automotive domain AUTOSAR operating system[13] is one of the most prominent architectures utilized. AUTOSAR employs a Real Time Operating System (RTOS) based on the OSEK standard [19]. This incorporates fixed priority scheduling. Offline before system start, mapping is performed for all tasks in the system. When the mapping has been finalized the scheduler can select the tasks to execute [20].

Another approach to the system handling is taken in the Rubus RTOS (RUBUS) [21]. This RTOS has the possibility to schedule its tasks with two different approaches. The system is separated into three distinct kernels called red, green and blue. The green and the blue kernels are both event triggered. While the Red kernel executes an offline pregenerated schedule. Typically applications that run this operating system allocate most of the program to utilize the Red kernel. This is done because it can provide a better software development process, since it increases predictability and eases verification. Rubus is among others used by Volvo Construction Equipment, so the system and its functionality is thoroughly tested. In regards to the approach used in this project the red kernel of the Rubus RTOS is quite similar since it uses offline scheduling.

An approach to handle motion control in automotive vehicles is presented in [16]. The method involves utilizing offline scheduling for the base of the system. Subsequently this is combined with an approach to online interrupt handling. The tasks are divided into different chains of a multitude of tasks. Where the construction of the schedule will take into account online interrupts, by allocating timing in the task chains.

Work is also being done with focus on utilizing multi-core solutions in the automotive industry [17][22][23]. In terms of scheduling, in these approaches the tasks have a need to be distributed over the different cores. Dividing the execution of tasks over several cores can provide an increased schedulability, since more tasks can execute at the same time. Then the tasks can be handled on each of the cores individually with the chosen approach. The work presented in this thesis entails scheduling jobs on a singe-core system. Although the approach could potentially be utilized even if a multi-core system were to be used. Consider static partitioning of the tasks over the cores, an offline schedule could potentially be generated for each of the cores. Thereafter, each of the cores would include its own dispatcher to handle the generated schedule [17]. This of course makes it more difficult to construct the schedule and divide the tasks since there might be more things to consider in regards to different conditions between different sets of tasks. Especially when considering potential end-to-end timing requirements that may exist in a given system. If that is the case, dividing the tasks over the multiple cores could become somewhat more complicated.

## 3.2 Real-time Scheduling

In [11] an approach to handle a mixed set of tasks with combination of offline and online scheduling is presented. The task sets considered can possess a number of different requirements and constraints. In this approach periodic and sporadic tasks can be guaranteed offline. In the generated offline schedule any complex constraint is transformed into a schedule containing simple start times and deadlines scheduled by EDF. At runtime an online algorithm is used to reduce pessimism and makes it possible to schedule event triggered tasks in combination with the periodic tasks. This work extends earlier work performed by the same authors [24]. Where only offline scheduling is used to handle the same type of situations. This shows how flexible it can be to create offline scheduling solutions.

## 3.3 Constraint Programming based Scheduling

Work has also been performed presenting scheduling approaches based on constraint programming [25] [26] [27] [28]. This is relevant to the work performed in this thesis, since it utilizes CP to help define the scheduling problem. In [29] an approach to generate offline schedules for tasks and messages to be transmitted over a bus, with the help of CP is presented. The work performed in [30] presents one approach where periodic tasks are offline scheduled based on CP. Although this was implemented for a multi-core architecture, in contrast to the work presented in this report which is amid for single core hardware. Nevertheless, implementation wise for each core it is similar to how the problem is tackled in this work.

Further, another work presented in [31] is relevant to this thesis. They propose a method to schedule tasks in a system with the help of Constraint Programming and more specifically the notation of Conditional Time Intervals. Further, they also utilize the IBM ILOG CP Optimizer in order formulate the scheduling utilize these timed intervals. This is similar to the approach proposed in this work, but in this work job-level dependencies are also taken into consideration when the scheduling problem is defined.

## 3.4 Discussion

Offline scheduling in general is not new and has been used in practice for an extended time period. This also holds true for approaches used in the automotive domain and other safety critical systems, where it is extremely important that assurance can be made that the system executes and behaves as intended. Further, the use of CP to solve scheduling problems has also been performed before. Providing solid ground for the intended approach of using CP in addition to its associated Conditional Time Intervals.

The new aspect considered in this thesis, that is the major contribution to the field, is to utilize job-level dependencies and apply them in an offline scheduling approach using CP. Tackling the problem of meeting end-to-end timing constraints, has not been handled in the intended way before. The concept of job-level dependencies themselves are a fairly new addition to the field on their own, therefore attempts to perform offline scheduling utilizing them have not been tested. Hence CP itself is also unutilized in this area. Subsequently, evaluating the effeicency using CP in combination with job-level dependencies to produce offline schedules with the goal of meeting end-to-end timing requirements, represent the finial contribution to the field of study. Lastly, considering this work focuses on simple systems using a single-core approach, the thesis can produce a solid ground for continued work. This includes adapting a similar approach to the modern multi-core approach.

# 4 Method

In this section two major things will be assessed. Firstly, the chosen scientific method will be presented, with a discussion on why and how it applies to the work performed in this thesis. Secondly, an overview of the factual approach to the problem will also be provided. This includes more detail on how the problem will be tackled and solved with some descriptions of the different aspect that construct the entire work of the thesis.

## 4.1 Research Method

For this thesis the majority of the aspects and theoretical approaches have been confined since the initialization of the work. Considering this, less time is needed to focus on finding an approach and more how to correctly use it given the definitions of the work. Some aspects of the work may need refinement or change during the process of the work. Because of this, a method that allows for improvements of earlier parts of the work based on findings during the later stages, would be very useful.

Considering this work should result in a functioning system, a good method to obtaining this is described by Nunamaker in [32] and can be observed in Figure 5. As can be seen in the picture it describes an multimethodological approach to research. It consist of four main categories: Theory building, system development, observation and experimentation. For this work the system development is in the centre of the work, this woks well with the method since the system development is also the centre of this method. Further the approach can be seen as an iterative process to the system development Thus, during any stage of the development things can be assessed and improved, based on the result of the succeeding steps. If backtracking is necessary to address an issue that may have been discovered, it is allowed. In accordance with the work performed in this thesis this is a good approach. Regarding the amount of steps in this work to arrive at a acceptable conclusion, the possibility to backtrack and improve various steps is extremely helpful.

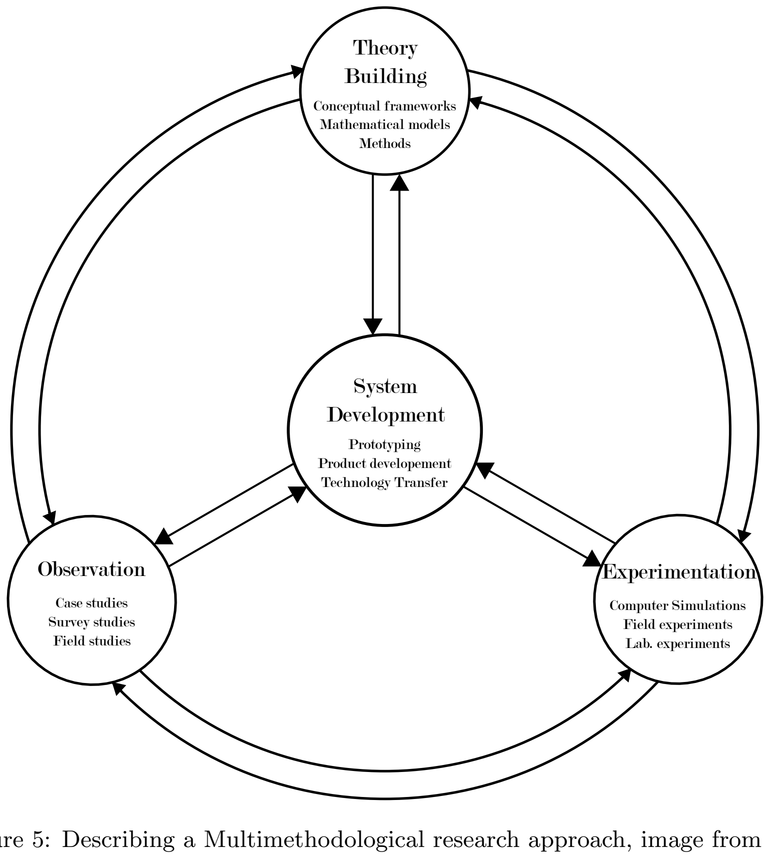


Figure 5: Describing a Multimethodological research approach, image from [32].

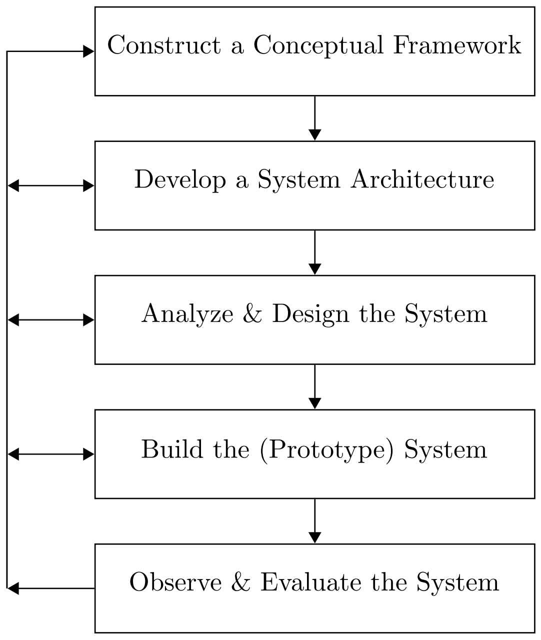

Construct a Conceptual Framework
Develop a System Architecture
Analyze & Design the System
Build the (Prototype) System
Observe & Evaluate the System

Figure 6: A Process for Systems Development Research, image from [32].

### 4.1.1 Application of the Research Method

The selected approach describes a systems development research methodology [32] that can be observed in Figure 6. Different questions and points that needs to be assessed for the work is presented in the figure.

How these steps are applied for this work is discussed below.

**Construct a Conceptual Framework.** The process of formulating and addressing the main research questions of the work. Assess in what way this will benefit and increase the knowledge in the research community, how it separates itself from similar work that has been conducted earlier.

Regarding the work presented in this report these questions have been formulated and presented in the first section. How scheduling of specific types of task sets given a particular setting and conditions, with the help of an offline scheduling approach is valid research based on the defined constraints.

**Develop a System Architecture.** Given the constraints and requirements, defined by the research questions and aim of the work, and subsequently design a solution based on this. What components the system requires to perform its task and the kind of functionalities needed in regards to the defined goals are assessed.

In this work some of these aspects are set based on the initial formulation of the problem. Although there are some points that still need to be decided, for the entire system to be implementable.

**Analyze and design the system.** When designing the system it is important that the information about the studied domain is known. Otherwise overlooking some unconsidered aspects could cause problems when trying to eventually implement the system. In regards to this work some major decisions should be made here how to exactly perform the scheduling and how the generated schedule should be defined so that it can be interoperated by the execution framework on the hardware. Considering the latter decisions need to be made as to how the generated schedule should be executed on the hardware.

**Build the system.** It is vital to get information how the system would actually perform in a real environment. For this to be done the system needs to be realised. Implementation of the system will also provide insight to the feasibility of the approach and how the functions perform. Further, it will provide information of the advantages and disadvantages of the concept, framework and the design.

In regards to the project presented, this part is important, since to fully evaluate how a schedule for a task set performs it needs to be tested. This will give good insight as to how well the approach actually works. Also discover possible weaknesses and non-functional aspects.

**Experiment, Observe, and Evaluate the System.** With a developed system at hand tests can be performed on it to evaluate various aspects. This can provide valuable insight as to how well the approach works and how it relates to the original research questions that desired an answer. In earlier part of the report some major things that are considered for evaluation is mentioned.

# 5 Technical Approach

This project consist of a number of interconnected parts that formulate the entire process of this work. In Figure 7 a simplified overview of the process can be observed. The first step in the process is to define the task set that will be the subject for scheduling. Further, the next step would be to define job-level dependencies that cover potential task-chains in the system model. Obtaining them will be done utilizing the same method described and used in the MECHAniSer tool [1]. Afterwards, when the entire set of tasks has been generated for a specific system, the need to schedule these arise.

Formulation of the offline schedule will be done utilizing CP and its conditional time intervals [8][9]. This will be used to define a scheduling problem. The CP will be defined so it covers all aspects in the system, from period deadlines to job-level dependencies. Construction and solving of the CP scheduling problem will be performed using the IBM ILOG CP solver [7]. This is a state of the art CP solver that also contains support for conditional time intervals. After the schedule has been created it will be forwarded to a real hardware platform for execution and evaluation.

Implementation of the execution framework is done utilizing the C language. The framework will be deployed to an Atmel AVR32UC3A [2] microcontroller. Further, to make the implementation straightforward, FreeRTOS [33] is used as a starting point. Otherwise implementing an entire operating system from scratch would be out of scope for this work. The provided schedule is represented by a C language h-file, outputted from the previous stage. This will be interpreted, subsequently FreeRTOS tasks can be generated and executed according to the timing obtained from the offline schedule. This is the system dispatcher and will be represented by a single task that will dispatch all the other tasks.

Thereupon, when the described parts have been implemented, so that a schedule can be generated and exported for execution on a hardware platform, evaluation becomes possible. Several different aspects could be subject to evaluation. Firstly demonstrating effects of the job-level dependencies on the end-to-end delay. Considering the amount of focus this thesis has on the utilization of the job-level dependencies, this will be an interesting part of the evaluation. How much improvement that can be observed in terms of meeting end-to-end timing constraints. Its also interesting to analyse the potential increase in solve times when utilizing dependencies. Moreover, performing a comparison against a generated task set without the dependencies is possible. This can provide an indication on the possible improvement using job-level dependencies in the schedule generation. Additionally, a "worst case" schedule could also be generated by changing the constraints, execute it on hardware and observe the the behaviour. Also the generation of the offline schedule could also be tested. Performing tests on randomly generated task sets could generate a benchmark against different characteristics such as scalability and schedulability ratio. In regards to the number of tasks in the set and the job-level dependencies. Further, most of the experiments can be performed on the schedule generation part, since it is performed offline. Additional smaller experiments could be performed on the hardware platform. Although, with the goal mostly being to demonstrate that a schedule is runnable, more than any type of benchmark testing. Hence, only simpler system models will be tested on the hardware. The more complex system model experimentation, will be mostly focused on the schedule generation itself.

Ultimately this should provide an answer if utilizing offline scheduling can generate schedules for task sets using job level dependencies. Both in terms of efficiency in regards to generating the schedules, and their final performance on actual hardware. In the end it will show how feasible this approach could be to problems of this variety. The approach derived from this work can potentially be used as a benchmark against future implementations using the same concept.

## 5.1 MECHAniSer

In [4], the MECHAniSer tool is described that presents an approach to aid with meeting end-to-end timing requirements. The approach used in this tool analyse sets of tasks taking into consideration so called cause-effect chains. The approach works by analysing a system at a high abstraction levels by looking at the set data age constraint for a given chain. If the timing requirements

[1]MECHAniSer tool information and download: www.mechaniser.com
[2]Atmel AVR32UC3A: http://www.atmel.com/products/microcontrollers/avr/32-bitavruc3.aspx

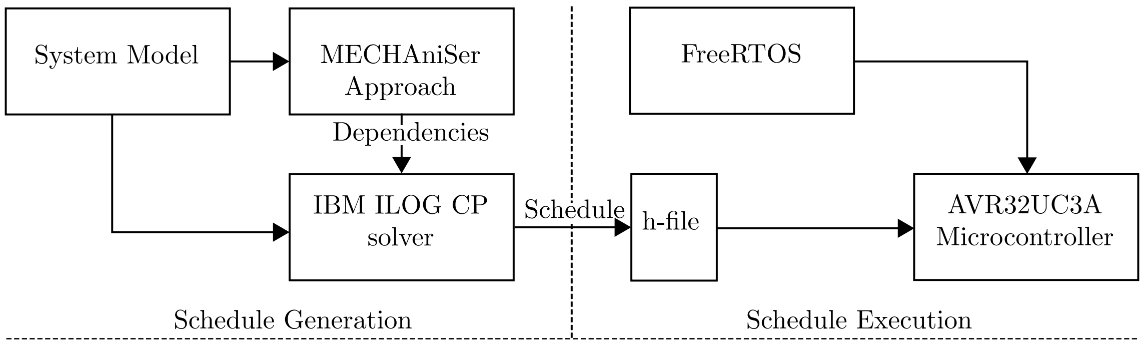


Figure 7: General overview of the project procedure.

are not met already the system model is augmented with job-level dependencies that should be fulfilled if the timing requirements are to be met. Making sure that data propagates thorough a chain of tasks within the defined end-to-end timing requirements, is the goal when utilizing the dependencies. Utilizing the provided job-level dependencies in subsequent schedule generation will guarantee end-to-end timing requirements are met if a schedule can be found. Utilization of the approach used in the tool is important since it lays out the foundation for the work presented in this thesis.

## 5.2 IBM ILOG CP Optimizer

Another vital part of the schedule generation will be the utilization of the IBM ILOG CP Optimizer [7]. This is a state of the art CP solver that provides support for conditional time intervals. The tool will be used to solve the scheduling problem defined using the conditions of CP and the conditional time intervals. It will be formulated based on the task sets provided from the MECHAniSer tool. When the problem has been formulated with all its conditions determined and set suitable solutions can be provided by the solver. When a solution that fulfils all constraints has been found this will be formatted in a suitable way for exportation to the execution framework located on real hardware. Generating a C language h-file will be done that should contain the calculated schedule. Considering the hardware implementation will be performed in C, having the calculated schedule provided in an associated h-file makes it simple for the program to interpret. In contrast, the previous step generating the schedule will be done using Java. Support exist for using the ILOG CP Optimizer with Java.

## 5.3 Atmel AVR32UC3A

Regarding the hardware platform, an Atmel AVR32UC3A [34] will be used for the validation of the generated schedule. This is a microcontroller supporting a 32 bit CPU running at frequencies up to 66 MHz. Further, the model used utilizes a 512 Kbytes flash memory. In some cases running an offline schedule on a platform with this amount of memory could pose a problem [35], depending on the size of the schedule. Although, for this work the space of the software will not be large enough, since it will mostly be a skeleton executing empty tasks to evaluate their performance. Lastly, developing and deploying for the hardware is made decently uncomplicated with the use of Atmel corporations own Integrated Development Environment (IDE) [3]. Additionally, since this IDE is developed with their own hardware in mind good synergy exist between them.

### 5.3.1 Execution Framework

For the purpose of evaluating the generated schedules an execution framework will be developed to run on the hardware. Thus, the requirement placed on the framework is that it should be able to interoperate the schedule in the form that it has been generated from previous stages. Thereafter it should be able to schedule tasks according to the provided instructions. Considering that fully

[3] Atmel Studio: http://www.atmel.com/microsite/atmel-studio/

implementing an OS from the ground is unnecessary, FreeRTOS [33] will be used as a base to construct the operating system. Utilizing FreeRTOS will make it easier to implement the system then it would otherwise. Further, the system needs to keep track of the results of the execution store them, and finally output them, possibly using a simple USART protocol. Lastly, development of the framework will be done using C.

# 6 Technical Description

In this section a technical description will be provided in regards to the offline scheduling approach. This covers all necessary aspects needed to replicate the scheduling process used to obtain the results presented in this work. Further, a small description is provided how the schedule was adapted to execute on the selected hardware. Although, this section can be easily adapted and changed to fit other hardware. This also includes the dispatch table generation, since it can be adapted to be generated in another way. But it should be easy to utilize in any system built upon using Free RTOS.

## 6.1 System Model

This section aims to provide information on how the system is described in the text. The system is represented by tasks, chains of tasks and job-level dependencies defined over specific sets of tasks. Each of these three parts will be presented in the following sections.

### 6.1.1 Task model

Common in automotive applications is the use of periodic tasks. In this work the tasks are defined using the same model. Generally tasks can be either time triggered or event triggered. Considering this project, all the tasks in the system will be time triggered, hence no consideration is given to tasks falling into the event activated category. As mentioned earlier, in Section 2.3, a task is defined by three major variables that decides how the task is allowed to operate. Describing the activation period, execution time and the time each task needs to have completed its execution.

Subsequently, in this work a single task $\tau_i$ is described using a tuple $\{T_i, C_i\}$. Where $T_i$ represents the period of a task, or the constant activation time between each instance of a task. Additionally, $C_i$ represents the WCET of a task. Further, deadlines in this work are assumed to be implicit, meaning that task deadlines are the same as the period $D_i = T_i$. Further, individual tasks will consist of several different jobs or instances denoted by $\tau_{i,j}$ wherein $j$ represents the instance of a specific task $\tau_i$. Additionally, each task instance $\tau_{i,j}$ is represented by the tuple $\{r_{i,j}, s_{i,j}, d_{i,j}\}$. The variables represent various timing values for each individual instance of a task. Where $r_{i,j}$ represent the release time of the instance and $d_{i,j}$ the deadline respectively. Hence, these two values describe the bounds for each instance $\tau_{i,j}$, where its interval should be of the same length as $T_i$ . Within this interval each $\tau_{i,j}$ needs to delegate its WCET $C_i$, where the start time is defined by $s_{i,j}$. Obtaining values for this variable will serve as the main objective of the scheduling problem. Furthermore, a system generally consist of several different tasks, all the tasks in the system belong to the set $\Gamma$. In Figure 8 the task representation described can be visually observed. Lastly, it is important to understand all task execution in this work is performed non-preemptively. Meaning once a task has started its execution, it must finish before any other task can be selected to run.

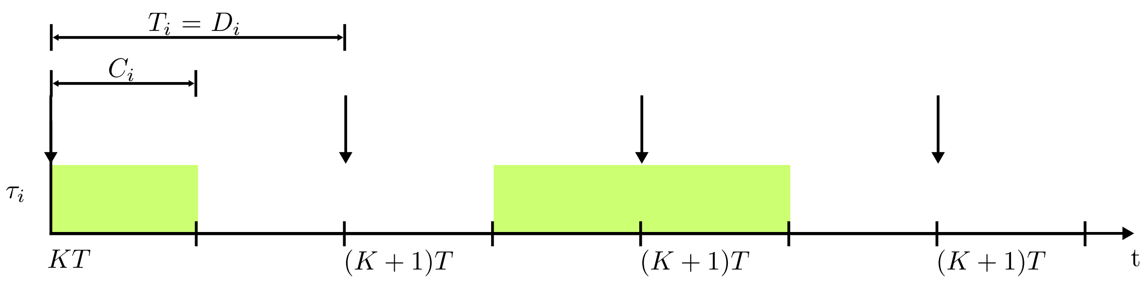


Figure 8: Showing an periodic task $\tau_i$ along with its associated period $T_i$, deadline $D_i$ and WCET $C_i$. In this example the deadline is the same as the period.

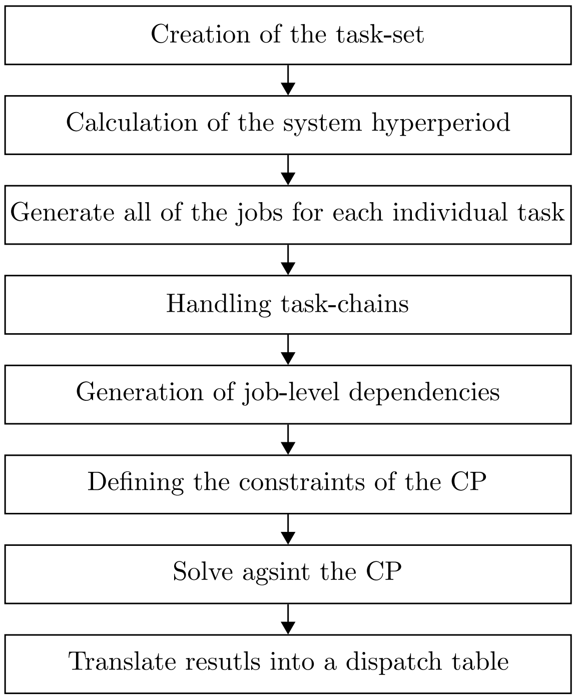

Creation of the task-set
Calculation of the system hyperperiod
Generate all of the jobs for each individual task
Handling task-chains
Generation of job-level dependencies
Defining the constraints of the CP
Solve agsint the CP
Translate resutls into a dispatch table

Figure 9: Flow chart representing the necessary steps in the generation of a valid schedule, from initial set of tasks to a dispatch table ready to be executed on a real embedded system.

### 6.1.2 Task-Chain Model

In the system a task chain is defined over a number of different tasks $\tau_i$ belonging to the set $\Gamma$ containing all the tasks in the system. A task-chain is used to describe data propagation through the system between tasks. Each of these chains has a defined timing constraint that dictates the maximum age that any data that is passed through the chain can attain before it is considered outdated and unusable. A specific chain $\zeta$ is presented in this work by another tuple $\{\nu, \eta\}$. Wherein $\nu$ is the representation of the tasks that belong to the specific chain, where $\nu \subset \Gamma$. The order of the chain is dictated by the ordering of the tasks in $\nu$. Any task in the system can be part of any given chain, and positioned at any position within it. A task can only appear once for every individual chain. Although, all tasks can be part of a multitude of chains at the same time. Further, $\eta$ represents the maximum allowed age of the data for the specific task-chain. All data passed through the tasks in the given chains use a *read-execute-write* approach. Meaning that the first priority of each task is to read possible data, perform an action and finally write the result to store it so the next possible task in the chain can read it. This dictates how and when the data will be available in the system. Further the time this takes also dictates what the end-to-end delay will be, which is never allowed to exceed $\eta$.

### 6.1.3 Job-level Dependency Model

A job-level dependency $\Psi$ is represented in the following manner: $\tau_i \xrightarrow{(j,l)} \tau_k$ dictating that instance $j$ of task $i$ needs to be executed before instance $l$ of task $k$ is allowed to run. Hence the two tasks $\tau_i$ and $\tau_k$ are part of the same chain $\zeta$. Where the created job-level dependency $\Psi$ is defined to help ensure the end-to-end timing requirements for the chain $\zeta$. Note, that several dependencies may be generated to cover any single chain, depending on the chain structure. Furthermore, each job-level dependency $\Psi$ covers the Hyperperiod (HP) of $\tau_i$ and $\tau_k$. Meaning it potentially needs to be repeated, to cover the systems HP, when attempting schedule generation.

## 6.2 Offline Schedule Generation

The process consist of a few different steps that needs to be followed to obtain a schedule that is fully ready to be executed on a real platform. In Figure 9, a flow chart is displayed providing a view of the necessary steps that should be performed for the successful creation of a schedule. Following this process, a fully executable schedule will be generated that also meets the defined end-to-end delay constraints provided by all the chains $\zeta$ in the system. This section will discuss the technical approach used. In Appendix B the tool developed, utilizing the same method described in this section, can be seen.

### 6.2.1 Create Task-Set

Before any schedule can be generated the tasks part of $\Gamma$ needs to be defined. When producing a task set it can typically contain any reasonable amount of different tasks. One important aspect to consider when defining a task set is that the $T_i$ is defined so that the generation of the HP is simplified. Meaning that a good approach would be to utilize periods that are multiples of each other. Selecting reasonable period values will also ensure that a valid HP can be calculated. Also avoiding hyperperiod that reach very large values. Considering the size of the final schedule will be smaller if the value of the HP is smaller. Further, another aspect to take into consideration is that the $C_i$ is defined reasonably in correlation to the selected value for $T_i$. Hence, creating tasks with $C_i$ too large in regards to the periods used, obtaining feasible schedules could prove to be impossible. Therefore it is vital that the task sets created take the described aspects into consideration, since very small changes could result in an unschedulable system.

### 6.2.2 Calculate Hyperperiod

When creating an offline schedule the goal is to find an ordering of all the $\tau_{i,j}$. Considering the goal of a real time system is to execute continuously once it has been initiated, some aspects needs to be considered in regards to the generated schedule. Thus, the ordering of all $\tau_{i,j}$ needs to be done in such a way that it can be repeated any number of times and still ensuring the same execution pattern. Achieving this is done by producing an ordering of all $\tau_i$ such that it covers an entire HP. Because if the schedule is defined in that manner it can simply be repeated indefinitely. This also produces a schedule that is as short as it can be, reducing the amount of space required to execute it. The HP is defined as the Least Common Multiple (LCM) of all $T_i \in \Gamma$, thus defining the total execution time wherein all $\tau_{i,j}$ needs to be sectioned. This ensuring that the generated schedule can execute continuously, by repeating a single pattern, while maintaining the integrity of the system in regards to the timing requirements.

The calculation of the HP is performed as an iterative process, as can be seen in Algorithm 1. One condition that needs to be fulfilled if calculation of the HP should be doable, is that the number of tasks in the system is no less than two. Most likely this is a situation that probably never will be an issue. Afterwards the HP is obtained by calculating the total LCM for all $T_i \in \Gamma$. During the first pass of the calculation process LCM is calculated for the first two tasks in the system. This provides the initial HP value that will be used in the continuing calculation process. Next the remaining tasks in the system are iterated through and the current HP value is continuously updated by using it as part of the LCM calculation together with the current selected task. Doing this ensures that the final obtained HP value is representative of all $T_i \in \Gamma$, meaning that the HP is dividable by all task periods in the system. As stated earlier this is important to ensure that the final schedule, covering the calculated HP, is repeatable during runtime.

Regarding the calculation of the LCM it is performed in two steps, described in Algorithm 2. Firstly the Greatest Common Divider (GCD) needs to be obtained. Acquisition of this number is performed by utilizing Euclid's algorithm. An iterative process that given two numbers identifies their GCD. Subsequently, when the GCD is obtainable calculating the LCM is possible. The process is simply the product of the two numbers divided by their GCD. This calculation results in the LCM that is the same as the hyperperiod.

It is important that the final HP calculated is correct for each of the tasks, otherwise the generation of each task instance later on will not be correct. Since the number of individual task jobs created is dependant on the HP. Without accounting for it creating a repeatable schedule would face

**Algorithm 1** Hyperperiod calculation

```
procedure CalculateHyperPeriod($\Gamma$)
    if $|\Gamma| < 2$ then
        return                                  ▷ Cant calculate LCM with less than two variables
    $HP \leftarrow LCM(T_0, T_1)$
    for $\forall \tau_i \in \Gamma, i < |\Gamma| - 1$ do
        $HP \leftarrow LCM(HP, T_i \in \tau_i)$
    return $HP$
```

**Algorithm 2** Calculation of the LCM

```
procedure GCD($a$, $b$)
    while $b > 0$ do
        $t \leftarrow b$
        $b \leftarrow a \mod b$
        $a \leftarrow t$
    return $b$
procedure LCM($a$, $b$)
    return $abs(a \cdot b)/GCD(a, b)$
```

potential difficulties. The goal is for the schedule to be as minimalistic as possible. A smaller schedule is both easier to initially calculate and subsequently easier to execute on a hardware platform. Further the amount of memory it requires also decreases, since the same pattern can simply be repeated ceaselessly.

### 6.2.3 Generate Task Jobs

Before any scheduling can be attempted each individual job or instance of every $\tau_i \in \Gamma$ needs to be determined. Considering the purpose of the scheduling is to assign start times for each generated instance $\tau_{i,j}$, that confines to the individual release and deadlines. Hence, the generation of each instance needs to be performed in accordance to the parameters that define each task. Considering that the final schedule must be able to run continuously without fail, all instances for each given task needs to be generated in a manner that enables the sequence to be repeatable. Utilizing the HP to determine the number of necessary instances will ensure that all instances will line up in accordance to their period when it reaches the value of the HP. As stated earlier this allows for repetition of the schedule. Considering this thesis is aimed towards real-time systems, the provided schedule should technically be able to operate indefinitely as per definition

The method used for generation of the individual instances for each task $\tau_i$ can be seen in Algorithm 3. First, the number of instances that should be generated for each $\tau_i$ is defined by $HP/T_i$. This calculation is performed for each task in the system. Considering the HP is obtained utilizing $T_i$ it will be evenly dividable for each value of $i$, producing an integer value for the number of instances $\tau_{i,j}$, as per definition of the HP. Thereafter, for each instance the release time $r_{i,j}$ and end time $d_{i,j}$ is calculated. The first value is obtained by multiplying $T_i$ with the current instance count $j$, while the latter is obtained by multiplying $T_i$ with $j + 1$. Performing these calculations will result in the individual release and deadlines for each task instance.
When the described process has been performed on each $\tau_i$ the basic system is ready to be scheduled, although without any regards for end-to-end timing requirements. Hence this lays out the foundation for the scheduling problem, along with defining the values for the scheduler to obtain $s_{i,j}$. To illustrate how task instances are defined, an example is presented in Figure 10, where a task $\tau_1 = \{4, 1\}$ is used to delineate the example. Where $\tau_{1,0}$ provide information how each instance $\tau_{i,j}$ is created, and within what confines it is allowed to operate. The time between each $s_{i,j}$ and $d_{i,j}$ is the same as $T_i$. Further, it is shown that the execution start time $s_{i,j}$ needs to be defined in between $r_{i,j}$ and $d_{i,j} - C_i$. Because the execution of a task instance cannot be initiated if the remaining time for the current task instance $\tau_{i,j}$ is less than the WCET $C_i$. When this process has been concluded for each task in the system all individual instances $\tau_{i,j}$ have been provided an

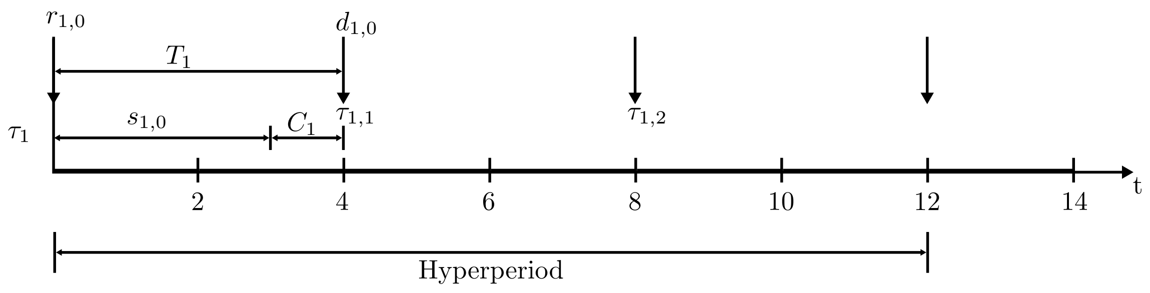


Figure 10: Representation of task instance generation showing one task $\tau_1$ which is a partial part of a bigger system.

**Algorithm 3** Generation of Task Instances

```
1: procedure GenerateAllTaskInstances($HP$)
2:     for $\forall \tau_i \in \Gamma$ do
3:         GenerateInstanceForIndividualTask(HP, $\tau_i$)
4: procedure GenerateInstanceForIndividualTask($HP$, $\tau_i$)
5:     $amount \leftarrow HP/T_i$                    ▷ The number of instances j
6:     for $j = 0, j < amount$ do
7:         $r_{i,j} \leftarrow T_i \cdot j$                  ▷ Release for each instance
8:         $d_{i,j} \leftarrow T_i \cdot (j+1)$              ▷ Deadline for each instance
9:         $j \leftarrow j + 1$
```

operational interval. Providing the start time $s_{i,j}$ inside the given interval is the unknown value in the scheduling equation.

### 6.2.4 Handling Task-Chains

For all chains $\zeta$ in the system a constraint $\eta$ is provided representing the maximum allowed data age for the given chain. If at any point during execution a task part of a chain exceeds its maximum data age $\eta$, the resulting schedule will not be valid. Despite the fact that a schedule could still be generated so that it is valid in regards to $T_i$ for all tasks in the system. Hence, for a schedule to be fully legitimate, meeting task periods as well as task-chain end-to-end timing requirements are equally important in terms of validity.

For any given chain $\zeta$ how the tasks $\nu$ are arranged, defines the order how data will be passed through the chain. Any task in the system can appear at any point in any given chain, additionally a task can only exist at one instance in any chain. Although, a task can appear in multiple chains at the same time. However, schedule generation increases in difficulty along with the number of chains any task belongs to at the same time, since more complex constraints are required. When constructing task-chains it is vital that realistic demands are set by the end-to-end timing constraints, to make it possible to successfully find a valid combination for a schedule. Many different aspects can define the difficulty to meet the demands of any given chain. This includes the number of tasks in the chain, more chains raise the difficulty to accurately fit all the task instances into correct time slots resulting in valid scheduling. Further, the task periods can cause difficulties as well, especially if the tasks in the chain do not share the same period. When they are different it leads to something called over and under-sampling situations. This can potentially cause big problems, therefore it is advisable to only allow tasks with certain period values to communicate with each other. Lastly the maximum data age value itself also needs to be reasonable if the given chain should be include able in a valid schedule. Generally it is good to provide a maximum data age that is greater than the HP of the tasks that compose the chain.

Calculation of the delay for each chain is necessary once a schedule have been generated. Analysing the actual maximum delay against the allowed delay $\eta$ will determine if the schedule is valid or not. Ensuring that all latencies in a chain are calculated correctly is performed for each

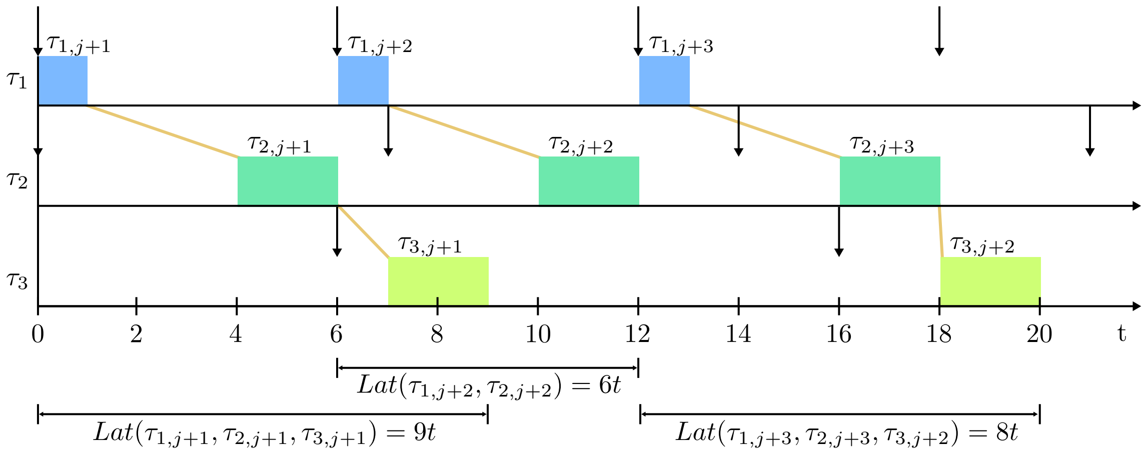


Figure 11: A partial trace of the tasks: $\tau_1$, $\tau_2$ and $\tau_3$ that are part of the same chain $\zeta$. The image displays how the delay is calculated on the propagation of data through the chain.

instance of the chain. A generated schedule for a finalized chain can be seen as a tree, where each node would represent the maximum delay for that local part of the chain. Starting from each node $\nu_{i,j}$ in the chain, delay is calculated from that point to the parent task instance. Traversing upwards in the chain consists of finding the closet task instance in $\nu_{i-1,j}$ until the top of the chain $\nu_0$ have been reached. In Figure 11 a partial snippet of a traced chain can be observed. It shows how the data progresses down the chain to different task instances, and how the delay is calculated. Calculation of the latencies is done for each complete propagation of data in the schedule, or until an "end node" is reached. In the partial trace shown in the figure three full propagations can be seen. Propagation of data from $\tau_{1,j+1} \rightarrow \tau_{2,j+1} \rightarrow \tau_{3,j+1}$ generates the largest delay of $t = 9$. In this partial part of the trace it would represent the maximum delay. Although, the preceding procedure would need to be applied in the same way for the entire chain to obtain the maximum delay. Hence, the calculation that yields the largest delay value will be the maximum delay for the chain. However, its important to note that the delay must be lower than the maximum allowed at any point in any given chain. Not just if the last task in the chain is reached, since any overstep of the timing constraint diminishes the validity of the schedule. Therefore, because of this reason the delay is calculated at each instance for the tasks $\nu$ belonging to any given chain $\zeta$.

### 6.2.5 Generation of Job-Level Dependencies

Central to this work is the so called job-level dependencies. Defined over the chains in the system to provide precedence constraints on the tasks that are part of any given chain. As mentioned earlier the approach to the generation of these dependencies are presented in the papers [3][4] that describe how these are generated and represented. All chains $\zeta$ in the system are analysed and all possible data propagation paths are considered, investigating if any exist that are confined within the maximum allowed data age $\eta$. If any path fulfil the requirements constraints are defined ensuring that the specific path will be utilized. This is what a job-level dependency is, a specification that a certain task instance must precede another one. Providing a limitation to the possible ordering of any $\tau_{i,j}$ part of any job-level dependency $\Psi$.

If it is not possible to generate any dependencies that will meet the maximum data age $\eta$ no arrangements can be made so that a valid schedule can be generated. Contrary, if dependencies are generated successfully the end-to-end delay of the chain will always be met, as long as the conditions defined are satisfied. Nevertheless, even if dependencies could be generated they provide no insurance that the system will be schedulable. It only ensures that the end-to-end constraint $\eta$ will be met if a valid schedule can be generated. The reason for this is that the generation of the job-level dependencies have no knowledge of the system or the scheduling approach that may be used. It only takes into consideration the provided task-chains and tasks and attempts to define conditions that will fulfil their requirements. When considering the definition of a job-level dependency another core could simply be added, so hypothetically every task would execute on its own core. Although this is not the case in reality, especially in this thesis where only single core solutions are considered. Because of this, situations may arise where dependencies are generated, but when accounting for them in the scheduling problem no solution can be found. Since additional requirements are placed on the scheduling problem all of them may not be fulfillable. Hence, causing the problem to be unschedulable.

### 6.2.6 Defining CP Constraints

Vital to the schedule generation is the specification of the various constraints that define the conditions of the . Accurately setting up the constraints over the system is important if a schedule should have the possibility to be generated in a correct manner. Defining these constraints will decide the rules and conditions that needs to be fulfilled when attempting to generate the schedule. What this process fundamentally entails is translation of the system model and job-level dependencies into the CP domain. Basically, obtaining a full representation of the system in CP terms, that subsequently can be solved for attempting to obtain a valid schedule. This section will cover the essential steps required to convert all necessary system parameters into the CP domain.

As mentioned earlier, to aid in the work with CP the IBM Cplex Solver will be utilized to define and solve the problem. Before anything else the timing variables in the system needs to scaled down. This is because the constraints needs to be defined with 32-bit signed integer variables whereas the timing variables in the system are represented by 64-bit signed integers. Therefore scaling the numbers down as much as possible allows for potentially smaller timing values to be utilized. Thus, the time variables used in the system are scaled down to be able to represent a greater range of time when defining the CP. The down scaling is described in Algorithm 4. The goal is to find the lowest number of trailing zeroes for any timing variable in the system. Based on this all timing variables can be scaled down by dividing them with $10^x$ where $x$ represent the lowest amount of trailing zeroes found. It is enough to simply check against all WCET values since these numbers will be the smallest numbers used in the system. All the timing variables in the system are scaled down to perform the CP calculations. Afterwards, when the problem have has solved, the answer generated will be recalculated to the original scale. One limitation is that if any WCET value is too low it may not be possible to scale down the system enough. Although, if micro seconds represents lowest allowed timing constant this should prove to be a non-issue.

**Algorithm 4** Calculation of a maximum divide amount

```
1: procedure GetMaximumDivideAmount()
2:     $C \leftarrow 18$                          ▷ Max amount of zeores for 64 bit int
3:     for $\forall \tau_i \in \Gamma$ do
4:         $C2 \leftarrow 0$
5:         $V \leftarrow C_i$
6:         while $V\%10 == 0$ do
7:             $C2 \leftarrow C2 + 1$
8:             $V \leftarrow V/10$
9:         if $C2 < C$ then
10:            $C \leftarrow C2$
     return $10^C$
```

The definition of the constraints can essentially be divided into two separate parts. Firstly, constraints need to be generated in a way so that the system would be schedulable, without any regards for the end-to-end timing requirements. Hence, this first part of the constraint definition will simply provide constraints with the capability to obtain a suitable schedule without any regard for end-to-end timing requirements. In other terms assuring that the individual periods $T_i$ are met for all tasks and instances. Therefore, the second part of the constraint definition is aimed towards providing suitable constraints that would ensure the system execution to be within all the given end-to-end timing requirements. Generation of the constraints aimed to meet the provided timing requirements, utilizes the previously generated job-level dependencies. As mentioned earlier, if a valid schedule can be found while simultaneously meeting all the restrictions provided by the

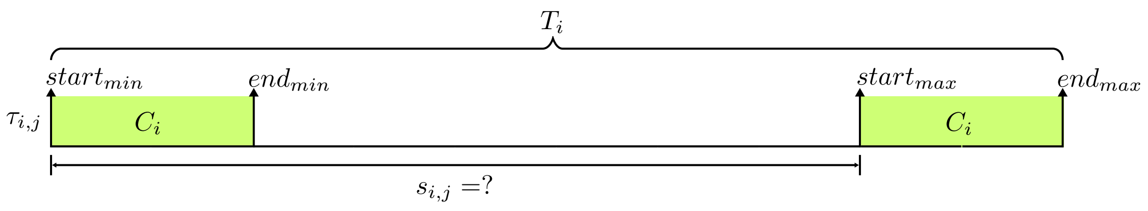


Figure 12: How the interval for each task instance $\tau_{i,j}$ needs to be defined. The $start_{min}$ and $start_{max}$ defines the interval wherein the execution of the task instance must begin. Consequently $end_{min}$ and $endmax$ defines the interval where the execution is allowed to end.

job-level dependencies, the system execution will always satisfy the end-to-end timing requirement.

Generation of the first basic set of constraints without regard for the end-to-end timing will be performed on every individual task instance $\tau_{i,j}$. For every single one a timed interval variable [8] will be defined. Described in an earlier step for each task instance $\tau_{i,j}$ a release time $r_{i,j}$ and a deadline $d_{i,j}$ was calculated. The goal of the scheduling is to define the value $s_{i,j}$ that represents the start time of the execution for the task instance $\tau_{i,j}$. The interval variable will represent the WCET $C_i$ that needs to be placed within the available timing window that each task instance has available, as can be seen in Figure 12. Each interval variable needs to be provided a clear interval wherein the execution start time is allowed to be assigned. Firstly a minimum and maximum start time is defined for the interval variable:

$$start_{min} = r_{i,j}$$
$$start_{max} = d_{i,j} - C_i$$

Where $start_{min}$ is simply defined by the release time for the current task instance $\tau_{i,j}$. In regards to $start_{max}$, it needs to be defined so that if the execution would start at that point it would finish precisely at the deadline $d_{i,j}$ of the current instance. Hence, the $start_{max}$ is equal to subtracting the WCET $C_i$ from the deadline for the task instance $d_{i,j}$. Consequently a minimum and maximum end time also have a need to be defined:

$$end_{min} = r_{i,j} + C_i$$
$$end_{max} = d_{i,j}$$

Where the $end_{min}$ needs to leave space for a possible execution that finishes at that point. Hence, the interval between the $start_{min}$ and the $end_{min}$ is equal to the WCET $C_i$. Further, the $end_{max}$ is merely defined by the task instance $\tau_{i,j}$ calculated deadline $d_{i,j}$. The defined interval constraints limit the interval that the CP can be solved for. This entails that the start time $s_{i,j}$ can only be defined in the interval between $start_{min}$ and $start_{max}$.

Subsequently once the interval variables have been fully defined for each job, the next step would be to add precedence constraints, as defined in the background Section 2.1.1. This would limit the possible ordering for each individual job, considering each job needs to finish executing before the next can start. Nevertheless, the current solution only accounts for task-sets utilizing implicit deadlines, rendering additional constraints redundant. The reason for this is that the interval variables have already been defined in a manner that fully limits the ordering of the jobs. The valid release times of succeeding jobs is always defined after the deadline of the previous. Hence, any additional constraint is not necessary when dealing with these types of systems, since the operational intervals have already been decided. Although, it might be necessary to reconsider the precedence constraints if other types of task-sets would be considered. In this thesis however, it is out of scope and have no need to be accounted for.

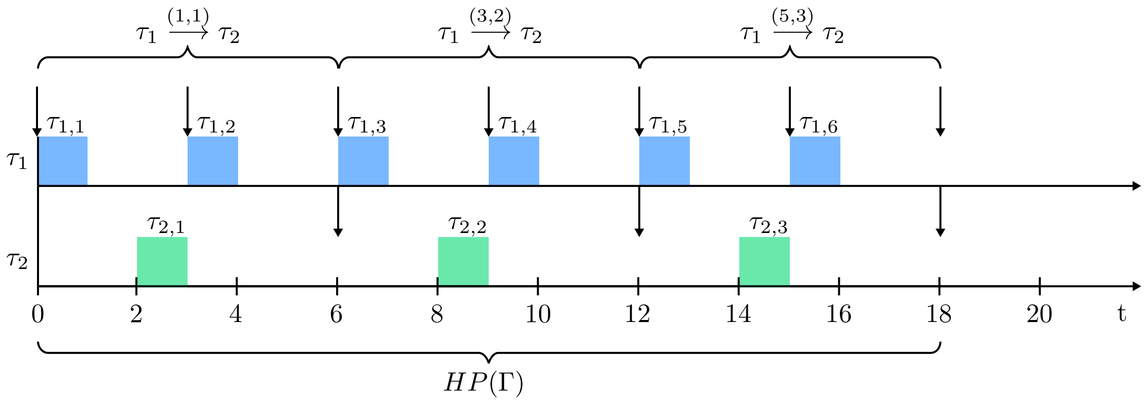


Figure 13: Showing how a constraint for a job-level dependency $\tau_1 \xrightarrow{(1,1)} \tau_2$ needs to be repeated to cover the entire system when the schedule is generated.

Certain conditions can be manipulated to attempt to solve the system in regards to different conditions. One such condition that always needs to be defined, when utilizing CP to define a schedule, dictates that there can be no overlap for any task instance in the system. Considering the generated should be used by a single core processor only one task instance can be executed at a single point in time. Hence, a condition of the solve is set to the following:

$$noOverlap(\tau_{i,j}) \quad forall \quad \tau_{i,j} \in \Gamma$$

The preceding condition applied to all task instances will ensure that they cannot be provided a start time located within another instances execution interval.

The constraints defined up to this point would be enough to solve for a schedule without regard for any end-to-end delay that could have been defined. Therefore, if the CP should attempt to solve for a schedule that include meeting the end-to-end timing requirements $\eta$ for each chain $\zeta$, additional precedence constraints need to be applied to the . Accomplishing this is done by utilizing the previously generated job-level dependencies. Stated earlier, it is a representation of what task instance $\tau_{i,j}$ needs to execute before another instance is allowed, if the end-to-end timing requirements should be guaranteed. Consequently, for each job-level dependency $\Psi$ that have been generated a number of precedence constraints needs to be defined. Consider the following dependency $\tau_i \xrightarrow{(j,l)} \tau_k$ stating that instance $j$ of task $\tau_i$ needs to finish its execution before instance $l$ of task $\tau_k$. Additional CP constraints representing the job-level dependencies can be defined as follows:

$$\forall\Psi \quad \forall x \in [1, \frac{HP(\Gamma)}{HP(\Psi)}] \quad endBeforeStart(\tau_{i,j} + ((x-1) \cdot \frac{HP(\Psi)}{T_i}), \tau_{k,l} + ((x-1) \cdot \frac{HP(\Psi)}{T_k}))$$

The preceding statement imposes additional constraints that needs to be satisfied to solve the . Hence, the precedence constraint added translates each job-level dependency in the system, to be part of the conditions of the scheduling problem. Meaning that the previous task $\tau_{i,j}$ must be fully processed before $\tau_{k,l}$ will have any possibility of doing the same. Additionally, precedence constraint will be applied in the same way for every job-level dependency $\Psi$ in the system.

Furthermore, as mentioned earlier, a job level dependency is generated to cover the HP($\Psi$), meaning the tasks that comprise the dependency. Therefore, in a situation where the HP($\Gamma$) differs from the HP($\Psi$), additional precedence constraints will be defined to fully cover $\Psi$. Constraints are added to cover the entire HP($\Gamma$), otherwise the constraints would not cover the entire schedule. Considering the HP($\Psi$) will always be a multiple of the system HP($\Gamma$), the number of times a constraint should be repeated will always be a whole number. Hence, the constraints can simply be repeated the number of times necessary and it will always fit the system. Further, when

repeating the constraint it is important that the correct step size is used to select the appropriate instances to add precedence constraints over. Therefore, in the equation the size is also calculated and it is multiplied by the number of times the constraint should be repeated.
When the step sizes and the repetition count is calculated the possible additional precedence constraints can be added, to cover the entire system. An example how constraints needs to be repeated can be seen in Figure 13. It depicts a partial system with a HP = 18 and two tasks $\tau_1 = \{3, 1\}$ and $\tau_2 = \{6, 1\}$. The two tasks are part of the same job-level dependency $\Psi$ with the following constraining factor $\tau_1 \overset{(1,1)}{\longrightarrow} \tau_2$. Further, the dependency has a $HP(\tau_1, \tau_2) = 6$ thereby giving a repetition count of $18/6 = 3$. Additionally, the step count for the task instance is calculated for $\tau_1$: $6/3 = 2$ and subsequently for $\tau_2$: $6/6 = 1$. Implying that precedence constraints needs to be generated over the following dependency requirements: $\tau_{1,1} \longrightarrow \tau_{2,1}$, $\tau_{1,3} \longrightarrow \tau_{2,2}$ and $\tau_{1,5} \longrightarrow \tau_{2,3}$. Approaching each job-level dependency in the same manner will ensure that the entire system is covered by the precedence constraints generated based on the dependencies.

### 6.2.7 Solving the CP

When all the precedence constraints have been defined the CSP is fully determined. Therefore, the next step is to attempt and satisfy the problem. If successful the solver will generate start times $s_{i,j}$ for every task instance $\tau_{i,j}$. When start points for every instance have been provided the schedule will be considered complete, since all aspects that define a schedule are known. Further, if the CSP took into consideration job-level dependencies and the schedule was generated it will have met all end-to-end timing requirements. Otherwise, if not considered it is possible that all end-to-end constraints are met but not ensured.

The solving of the CSP is fully performed by the ILOG CP solver. Further it is also possible to instruct the solver to try and maximize or minimize the solution in regards to the WCET $C_i$. Using a minimization problem would cause the solver to attempt and push the start times $s_{i,j}$ as close as possible to the release time $r_{i,j}$. Contrary, the maximization problem would attempt to push $s_{i,j}$ closer to the deadline $d_{i,j}$. Nevertheless, it is also possible to not define any minimum or maximization problem. In that case the first generated schedule that satisfy the CSP would be selected.

When solving against the constructed constraints the solver will attempt to explore the possible paths that exist. Depending on the number of jobs and constraints, the search space could turn out to be very large. Ipacting the difficulty in finding a solution, as well as increasing the solve time. Furthermore, depending on the preconditions of the solver the best possible path for the specific solve condition will be selected. Afterwards the calculated start times $s_{i,j}$ will be applied for every individual instance $\tau_{i,j}$ in the entire system $\Gamma$. Thus, the schedule has been generated and the process is complete. At this point all variables in the system are known, which is one of the preconditions for a offline schedule. Meaning that it is ready to be fully executed on a real platform.

### 6.2.8 Generate Dispatch Table

The dispatch table is generated so that it can be used to execute on a real hardware platform. Considering in this thesis the implementation of the execution framework was performed with the help of the C language, the dispatch table is represented by a C file along with a corresponding H file. These two files contain the generated schedule information so that it can be adapted and used together with the code that make up the execution framework. Where the most important is the start time for each task instance, the very number that is decided when the CP problem is solved. Each unique task in the system will have its information part of the generated table, occupying 16 bytes of memory. Further each individual task job in the schedule take up some 10 bytes of memory, storing the necessary information instructing when a given job should start execution. Considering no focus was placed on memory optimization in this work, since the goal was to provide a proof-of-concept for smaller system models, the amount of occupied memory could potentially be refined. Since the schedule is predefined it will be incorporated when compiling the code before deployment to the hardware platform. This is the final step before the entire schedule generation process is complete. Afterwards it is up to the execution framework itself to make sure that the

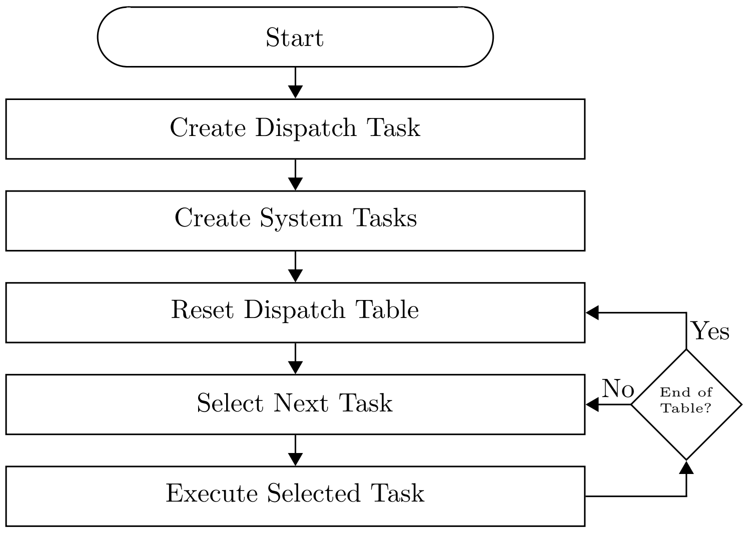


Figure 14: Simple flow chart showing how the execution framework operates.

schedule is executed correctly. Finally, an example of how a dispatch table can be formulated is observable in Appendix A.

## 6.3 Execution Framework

The purpose of the execution framework is to interoperate the schedule represented by the dispatch table, and subsequently distribute and run tasks accordingly. To avoid developing an entire operating system from the ground up FreeRTOS is utilized as mentioned earlier. How the schedule is interoperated and executed is outlined as can be seen by Figure 14. Where a dispatch task will be responsible for traversing the dispatch table and subsequently select the tasks to run at certain time intervals. Before the scheduling starts all tasks are defined. The tasks in the system are set to use the same priority, including the dispatcher task. Further, FreeRTOS is configured to use no preemption, therefore running tasks need to yield if a rescheduling should occur. Upon starting the scheduler all tasks except for the dispatcher are placed in suspended mode, meaning they cannot be selected for execution. Hence, when starting the scheduler the dispatcher task is the only available task to run.

All individual jobs in the dispatch table are ordered according to their release time. Hence, the dispatcher task will idle until the release time for the next task has been reached. Upon arriving at latest task release time it will be resumed. Subsequently, the dispatcher task will yield and since the only other active task in the system is the one selected to run no other task will resume executing. The selected task will execute for the length of its designated WCET, and afterwards yield. The dispatcher task will be resumed and suspend the last executed task. Afterwards, if additional tasks still need to be selected to run in the dispatch table a new task will be selected and the process repeated. Otherwise, if all tasks have been provided execution the process will start over again from the beginning of the dispatch table. Meaning that the pattern will repeat over and over until the system is shut down. This is all that is needed to run any of the created schedules based on the method described earlier.

# 7 Evaluation

Measuring the effectiveness of the presented approach can be done by observing a number of different aspects. Firstly, possibly the most interesting experiment to perform is in regards to the schedulability of the system, in addition the the amount of end-to-end deadlines that are met. Further, the time it takes to generate the schedules are also of interest. Especially in regards to the impact on performance when taking into account job-level dependencies. Both in the time it takes to generate the actual dependencies and subsequently solving for a valid schedule using CP. Another aspect to consider in terms of performance is what characteristic render the largest impact on the scheduling performance. Identifying this feature could potentially make it easier to extend, adapt and improve the approach to fit systems with other specifications.

Before any experiment can be performed, a system model needs to be generated that consists of all the necessary tasks and chains that makes up an entire system. Considering that the validity of the experiments are dependant on the generated test cases, it is important that the generation attempt to mimic reality and in this case automotive systems. Nevertheless, generation of the test cases needs to be performed randomly while still staying within certain defined aspects representing task-sets that could exist in real systems.

Representation along with the generation of the test cases are based on the total utilization of a system in regards to its associated tasks. The utilization represents the amount of CPU time that will be occupied by the tasks in the system. Where a hundred percent utilization mean that some task will always be executed, with no idle time. The utilization of a system is dependant on the number of tasks, how often they need to be executed and the WCET for each of the tasks. Thus, basing the generation of system models on utilization will provide results for different stages of complexity. Enabling observation on how performance develops along with the size of the system models to schedule.

## 7.1 Generation of Test Cases

With the intention to produce test results as accurate to the real world as possible, that can be comparable with real life situations, the generation of the task-sets used are very important. Considering this work is mostly aimed towards the automotive domain, the task-sets generated for the experiments attempt to mimic real automotive systems. Nevertheless, the great number of task-sets needed to produce accurate results and evaluation of the high amount of different situations that can arise, require the generation to be versatile. Meaning, that the randomness factor in the generation needs to be as large as it can, while still confined to the restrictions that is placed upon it. Regarding this work, generating task-sets that fulfil all these requirements in a satisfactory way is not entirely non-trivial.

Several conditions should be met during the creation of the test cases. As this work focuses on handling systems utilizing cause-effect chains, it presents the most difficult part of the model generation. Creating both random chains and tasks that match, while simultaneously producing a system that is random enough to represent the great number of situations that are possible, represents the main issue. Essentially, two initial steps needs to be considered, generation of the

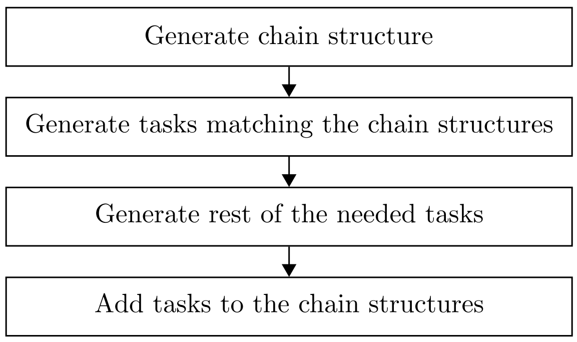

Generate chain structure
Generate tasks matching the chain structures
Generate rest of the needed tasks
Add tasks to the chain structures

Figure 15: General overview of the necessary steps to generate system model for a test case.

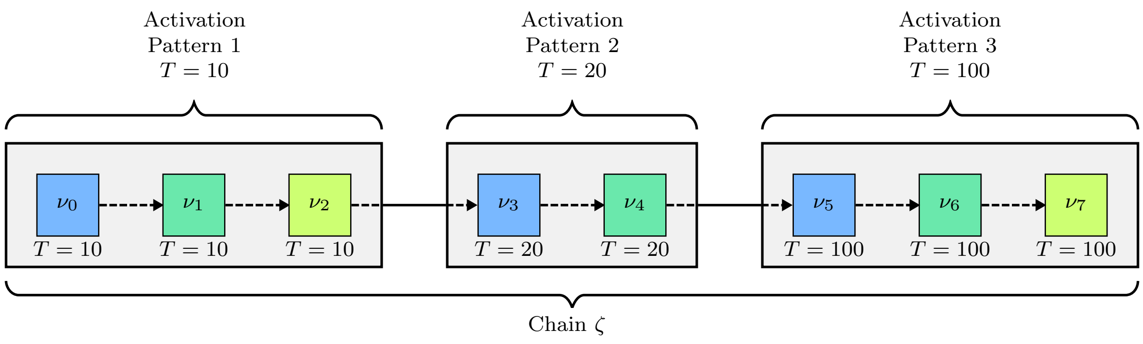


Figure 16: Depicting how a chain $\zeta$ can be generated for a test case. The displayed chain consist of three activation patterns with three, two and three runnables respectively defining a chain with a total of eight tasks.

| **Involved Activation Patterns** | **Share** |
|---|---|
| 1 | 70% |
| 2 | 20% |
| 3 | 10% |

Table 2: Number of activation patters used for each chain.

| **Number of Tasks** | **Share** |
|---|---|
| 2 | 30% |
| 3 | 40% |
| 4 | 20% |
| 5 | 10% |

Table 3: Number of tasks used for each activation pattern.

tasks and generation of the chains.

The process of generating a valid system model is presented by a flow chart viewable in Figure 15. To tackle the main issue of the model generation, the structure of the task chains are generated first. This ensures that the chains in the system will be random, and not forced to fit any given task-set. Generation of each individual chain follows requirement defined in [36], describing real-world applications. Based on the descriptions provided the method of task chain generation can be defined. It involves selecting a number of activation patterns and the number of tasks for each pattern, as can be seen in Table 2 and Table 3 respectively, to make up the chain. An activation pattern holds a number of tasks that all share the same period. It is the basic building block of the task chain generation. Following these numbers the maximum number of tasks that can exist in a chain is fifteen and the lowest is two. In Figure 16 an example of a chain can be seen and how it relates to the activation pattern. In the example the chain is generated by three activation patterns, consisting of three, two and three tasks respectively. Each activation pattern will have a designated period, given from the same paper [36], in the range of {1ms, 2ms, 5ms, 10ms, 20ms, 50ms, 100ms, 200ms, 1000ms}. All the tasks inside a pattern share the same period as the pattern itself. Lastly, subsequent activation patterns for one chain needs to be assigned a period with valid communication authority in regards to the previous activation pattern. The communication validation is based on the periods of the activation patterns, given from [36], and can be seen in Table 4. This table shows the task periods that are allowed to communicate between each other, to produce more operable systems. This process is repeated in the same way for the number of wanted chains in the system. Subsequently, deciding the maximum allowed data age for the individual chains is needed, representing the finalizing step in creating the chain structure. Maximum data age is generated by taking the hyperperiod for each chain and multiplying it with a multiplier in the range of {1.2, 2}. This gives a maximum data age that is at most twice the size of chains hyperperiod. Deciding this value is not too easy. Considering the number cannot be to large, since the deadline could always be met regardless of scheduling order. Further, if the value would be generated to low it could turn out to be impossible to fulfil. Hence, this number was selected mostly based on intuition, since no set number exist that inform what this value should be defined as.

| **Period** | 1 ms | 2 ms | 5 ms | 10 ms | 20 ms | 50 ms | 100 ms | 200 ms | 1000 ms |
|---|---|---|---|---|---|---|---|---|---|
| 1 ms | ▨ |  |  | ▨ | ▨ |  | ▨ |  |  |
| 2 ms |  | ▨ |  | ▨ | ▨ |  | ▨ |  |  |
| 5 ms |  | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |  |  |
| 10 ms | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |
| 20 ms | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |
| 50 ms |  |  | ▨ | ▨ | ▨ | ▨ | ▨ |  |  |
| 100 ms |  | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |
| 200 ms |  |  |  | ▨ | ▨ |  | ▨ | ▨ | ▨ |
| 1000 ms |  |  |  | ▨ | ▨ |  | ▨ | ▨ | ▨ |

Table 4: Allowed communications between tasks in the task chains. Each grey cell represents an allowed communication between two different period values.

Afterwards, once the chain structures has been generated, task-sets containing tasks that fit the chain structures is attempted to be generated. Ensuring that there will always exist some number of tasks that match the previously generated chain structures, the initial tasks will be forced to have a period that accommodates the chain structures. This implies that there will always be a number of tasks generated that will utilize a period that will fit the chain structure. All the other tasks will have their period randomized from the same set of periods used earlier. Obtaining the WCET for each task is randomly selected from a range of {80us, 200us}, that represents WCET commonly found in automotive systems [37]. For each test case, the number of tasks that will be generated is dependant on the desired utilization given for the current model. When a new task is generated, the utilization it produces is calculated akin to Equation 1, and subsequently tallied to a total utilization count. Hence, when the tally reaches the desired utilization the task-set is considered complete and task production is ceased.

$$U_i = \frac{C_i}{T_i} \tag{1}$$

Finally, the chains needs to be created based on the previously generated chain structure. Everything about all the chains is know except what tasks are part of what chain. The generated tasks ensure that at least one combination exist for each chain, although several different combinations are a possibility. Considering this, for each chain tasks are randomly selected from the produced task-set and attempted to be added to the chain. This process is repeated until all chains in the system have been assigned a full compliment of tasks, finalizing each chain. Randomly selecting from the entire task-set, not just the tasks crafted to ensure chain creation, makes the final test case more stochastic. Thus, it will not be too limited even though several limitations are imposed on the test case. Where the most important is that a certain utilization is achieved. Increasing the number of different situations covered by the test cases, produces results that provide more

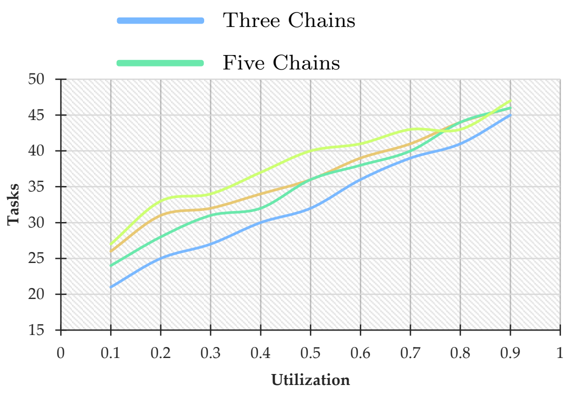


Figure 17: Showing how the number of tasks increases in an essentially linear manner. With the line being somewhat steeper when using more chains, but all converging towards the same number for the higher utilizations.

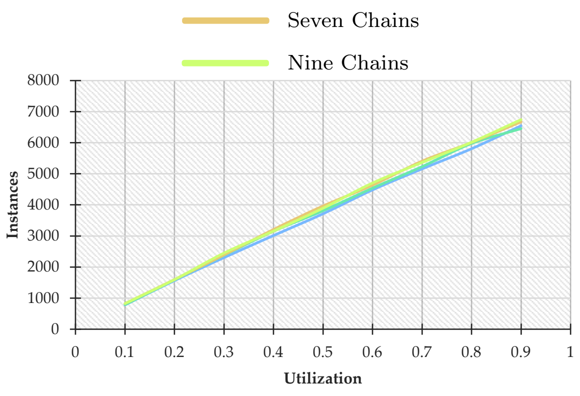


Figure 18: Showing the number of individual task instances that are generated when using different number of chains in the system generation. No matter the number of tasks the number of instances generated stays mostly the same.

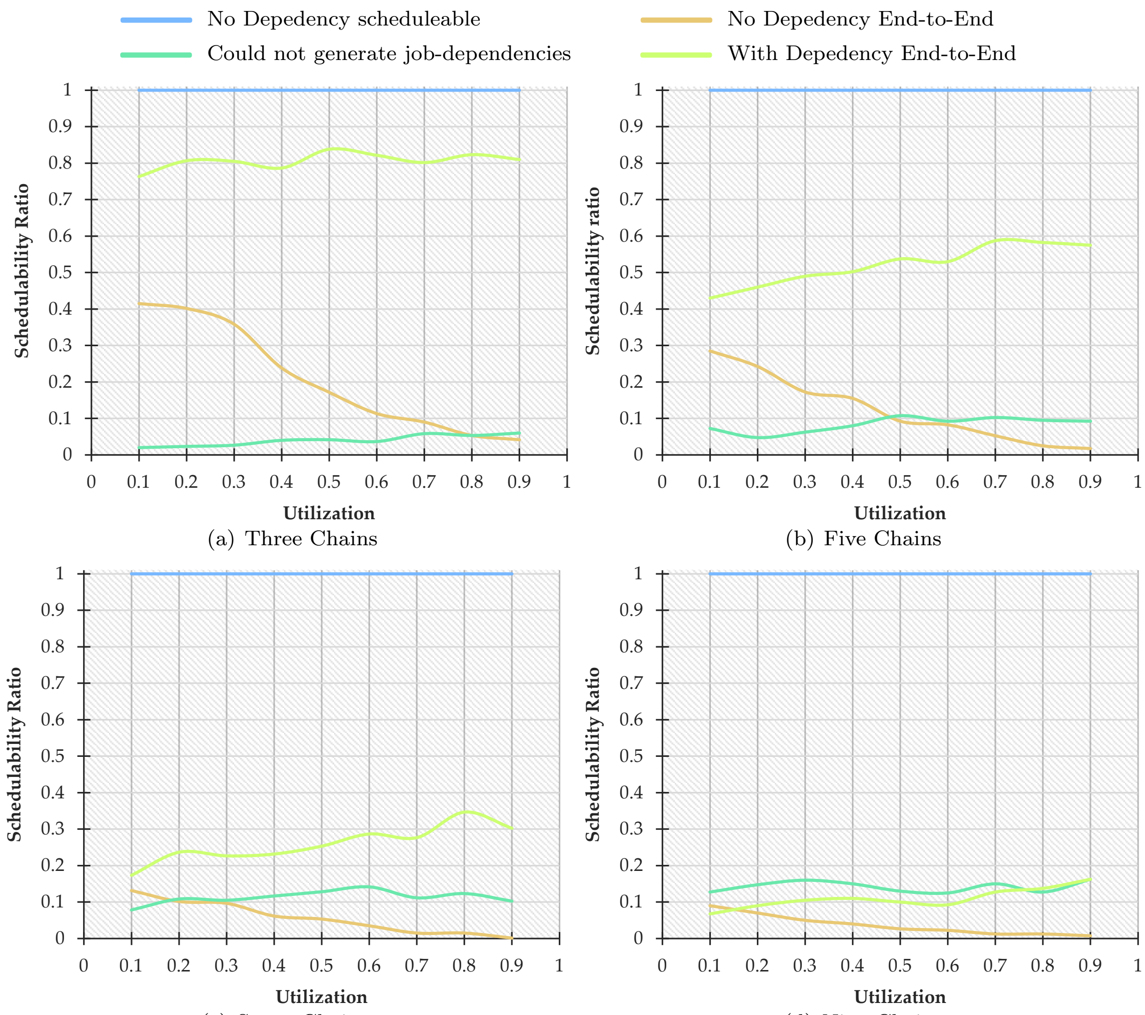
accuracy in regards to reality.

Importance is placed on the randomness of the system generation, while still assuring operation within acceptable parameters, especially how the task and job number change in accordance to the desired utilization. In Figure 17 and Figure 18, based on samples from around 200 iterations for each number of chains considered in the system model generation, it can be observed that the number of tasks and the number of jobs created for the different utilizations, increases approximately in a linear fashion. Showing that the systems generated grows evenly when increasing the wanted utilization. This despite the number of chains used when generating the system. Worth noting is that the line is steeper for the lower number of chains in regards to the tasks generated. This is because more tasks are needed when using a larger number of chains, therefore an additional number of tasks could be necessary. Although, the number of tasks still reach the same number when approaching the higher levels of utilization. Also, the number of instances for all the tasks stays approximately the same, regardless of the number of chains. Naturally, these are average numbers over several iterations and the number may vary to some degree for each individually generated system model. Nevertheless, overall system model generation behaves in a predictable way with an even increase when utilization is increased. Knowing how the number of jobs and tasks increase along with the utilization, makes it easier to draw conclusions in regards to schedulability and performance.

Lastly, it needs to be accounted for how the generation procedure can have a large impact on the experiments being performed. How the tasks are defined in regards to periods, WCET, the number of chains and the number of tasks for each chain, will have a big impact on the total number of tasks and instances generated. How congested a system turns out could have an impact on the possibility to meet the end-to-end delays. If most tasks in a system would be part of one chain or possibly more, it could make it difficult or impossible to meet some of the end-to-end timing requirements, especially when considering a single core approach. Therefore it is important to recognize the limitations of the generated system models when reviewing the results produced. It is also important to understand that randomly generating the system models, can produce systems that are inherently unschedulable.

## 7.2 Schedulability and End-to-End Timing Requirements

Measuring the efficiency in creating valid schedules and meeting the end-to-end delays, specified by the chains present in the system, is controlled by changing two variables when generating the test cases. Firstly the utilization, that will dictate the number of tasks and their characteristics. Secondly the number of task-chains that should be generated, depending on this number the performance can potentially change a great amount. Considering scheduling of systems with cause-effect chains is the main focus of this work, the experiment will be based on increasing the number



(a) Three Chains (b) Five Chains

(c) Seven Chains (d) Nine Chains

Figure 19: Displaying the success rate finding a valid schedule in addition the the number of instances all end-to-end requirements are met. Both aspects are displayed when job-level dependencies are in use as well as when they are not in use.

of chains created to accompany the task-sets for the randomly generated systems. Increasing the number of chains will greatly increase the difficulty to both schedule and meet the end-to-end requirements. One major aspect that must be accounted for in regards to the performed tests, is the characteristics of the job-level dependencies generated to meet end-to-end deadlines. Considering that the dependencies have no previous knowledge of the system or the scheduling approach, they may not always turn out to be fulfillable. Thus, when attempting to obtain a schedule for certain systems it may not work, even though the job-level dependencies could be fully generated. The major reason for this is that the work focuses on solving schedules using a single core approach. Further, the job-level dependencies assumes any number of cores are in use, because of this all dependencies may not be satisfiable when added to the . Therefore the produced experiments aims to identify the efficiency using a single core approach, in the attempt to solve these issues. Additionally, find the point where singe core scheduling starts failing to produce satisfactory results.

The experiment is formulated around randomly generating system models for a number of different utilizations, spanning from 10 to 90 percent. In addition, a set number of chains to be generated for each test case is also defined, across the different tests the number has been between zero to ten. For each utilization point the experiment is repeated several times, with the intention to produce a more accurate mean value of the results. For the following test 200-400 iterations

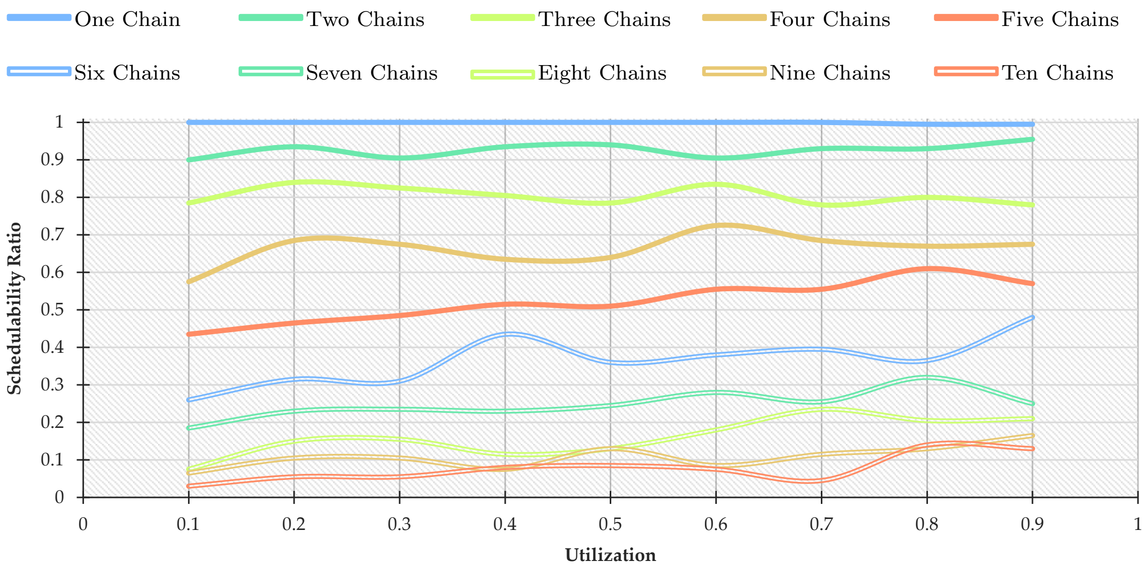


Figure 20: Showing the performance for a number of different chain amounts in regards to the end-to-end timing requirement. Showing that the result diminishes for each added chain to the system model. This behaviour is because task-chain congestion increases making it more difficult to create valid schedules.

were used for each point of utilization. The number is a bit different since some cases take a very long time to run compared to the others. Each time a new set of tasks and chains is generated, in accordance to the method described in Section 7.1. For each iteration and its generated system model a solving attempt is performed with both job-level dependencies considered and without. Hence, it will be possible to observe the differences when utilizing job-level dependencies opposed to not. Additionally, the attempted solving by the CP in this test do not use any minimization approach, it will use the first solution found, to optimize the run-times.

Selected results of the experiment can be viewed in Figure 19. Results for four different chain amounts present in the generated system model is described. The specific charts displayed have been selected to show the development of the results when additional chains have been added to the system model. Note, possible oscillation can be related to the number of iterations for each data point, meaning that an increased amount of iteration would smooth the lines. Time is a limited factor in this case, since running that number of tests requires a good amount of time. Further, the randomness of the system generation can also have an impact, some generations could present disproportionate difficulties in the generated models. For different lines can be observed in each of the presented charts. The results is presented in terms of schedulability ratio, meaning the amount of cases that resulted in valid schedules. Considering the aim of this thesis is to obtain valid schedules, the results presented are binary. Meaning, that either a system model will be schedulable or it will not. Therefore, based on several hundred iterations the schedulability ratio is obtained by classifying each system model as simply valid or not valid. In turn, this provides a percentage value of success, that is presented in the charts as schedulability ratio. The charts depict schedulability ratio in terms of meeting the end-to-end timing requirements, with dependencies considered and without. Also, the charts depict the schedulability ratio simply considering task periods and not the end-to-end timing requirements. Lastly, the amount of times where no job-level dependencies could be generated is also presented.

Observing the results it is notable that regardless of utilization a system will most likely be schedulable without regard for any end-to-end requirements. Regardless of the fact that the method used is non-preemptive. This might be because relatively low WCET are utilized, if higher would be used the result could turn out to be different. Further, satisfying the end-to-end timing requirements without dependencies is mostly based on luck and the results fall of quick when utilization is increased. This signifies the importance of utilizing job-level dependencies in the schedule generation. Answering research question 1(c), that the schedulability ratio when considering end-to-end timing requirements, increases significantly when dependencies are considered. Using job-level de-

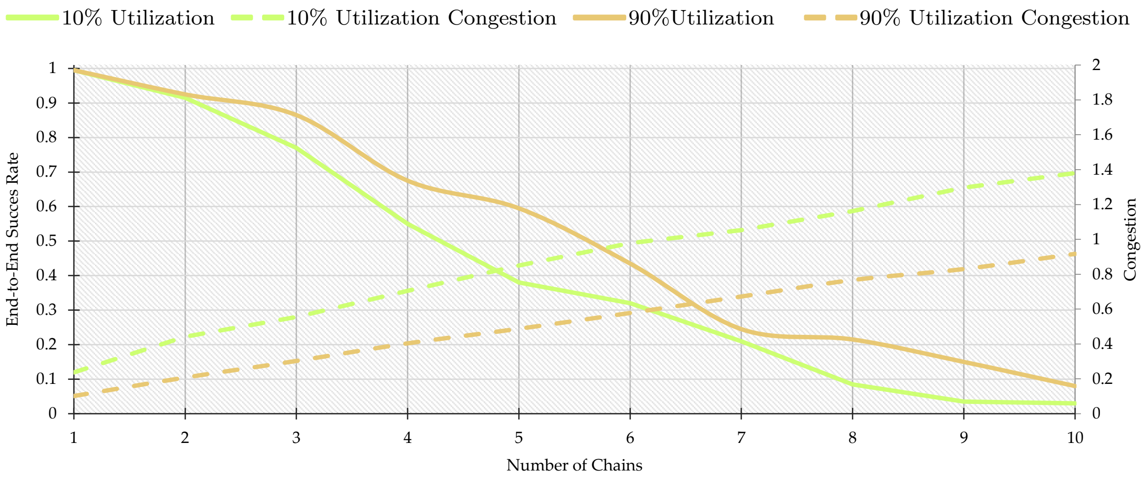


Figure 21: Showing how the end-to-end results decreases while the congestion increases. The low utilization have been generated with a ten percent utilization and the high utilization generated with 90 percent. System models have been generated from one to ten chains to plot the development when additional chains are added. Additionally the two dashed lines represent the congestion for the two solid lines.

pendencies in the generation of the schedule, will cause the system to be less schedulable when disregarding the end-to-end timing, this since they are bound together. Nevertheless the job-level dependencies achieve far better results in regards to the end-to-end requirements. The improvement is further facilitated when the utilization of the system increases. Without usage of the job-level dependencies the results fall off rapidly. The opposite holds true when using the dependencies in the schedule generation. The results stay mostly the same regardless of the utilization amount. Individual result in terms of end-to-end delay using job-level dependencies for different amount of chains and utilization, can be seen in Figure 20. It can be noted that the falloff in success is large when more chains are added to the system model. Further, it is interesting to observe that the result stays essentially the same regardless of the utilization for all chain amounts displayed. This provides an interesting answer to research question 1(b), in terms of schedulability ratio. That when the system increases in size, when considering the amount of jobs, the results do not fall off. Although as noted this does not hold true when adding additional chains. From this it can be argued that the results are closely tied to the generation of the system model, regarding the amount of tasks along with the number of chains created. If to few tasks are available for inclusion into several different chains, situations will arise where meeting the end-to-end seems impossible.

### 7.2.1 Task-Chain Congestion

Reasoning about the behaviour of the system in terms of reduction in schedulability when additional chains are added, and somewhat of an improvement when utilization increase, can be done while observing a systems task-chain congestion. A representation of the relationship between the number of tasks in the system and the number of chains. When a system consist of a lower number of tasks, the available options to populate task chains decrease. Therefore, when more chains are added, the congestion of tasks in regards to the number of chains increase, if the number of tasks stays fairly unchanged. Potentially this can have a big impact when attempting to schedule a given system model. Considering a case where the amount of congestion is large, it is possible that essentially all tasks in the system are part of a chain, and potentially several different chains. Situations falling into this category, could result in an unschedulable systems. Because the number of conditions needed to be fulfilled rises, and not all might be met. In the experiment presented in Figure 19, this is what is happening. When additional chains are added to the system generation, the results fall off. The number of tasks in relation to the number of wanted chains in the system, causes the success rate meeting the end-to-end requirements to decrease. Although, this is to be expected when the systems are created in a certain way, since all possibilities are limited to the conditions during the generation process of the system model. Also it can be seen that most of the results in regards to the schedulability ratio, increases when the utilization gets higher. The reason behind this is that the number of tasks increase while the number of chins stay the same, hence the congestion decrees.

An attempt to illustrate that, when the congestion increases by adding additional chains, the results will decrease along with the increase in congestion, as can be seen in Figure 21. In the test, system models have been generated with a utilization of ten and ninety percent, for a number of different chain amounts. It can be seen that the success rate meeting the End-to-End delay constraint decreases for each added chain. Further, it is worth noting that the task-chain congestion increases as well. Additionally, when more chains are created when the system have a higher utilization, the congestion is lower and the result improves. Even though the utilization should make the system harder to schedule, it still generated better result in this case for the end-to-end requirements. It might seem counter-intuitive, but when more tasks can be selected for the different chains, the overlap decreases. This results in constraints that can be increasingly solvable.

Congestion consists of two different aspects that attempt to describe the state of a system model, and is calculated as described in Algorithm 5. The first aspect provides information on the amount of overlap that exist in the generated system models. Where the number of tasks in any chain is calculated, along with the total number of tasks included in chains, also accounting for tasks part of several chains. Dividing these values will provide a number representing the amount of overlap for all the chains in a model. A value of one represents the lowest possible value and indicates there is no overlap, all chains consist of tasks that are part of no other chain. Where as bigger numbers indicates that tasks are part of several chains. The second representation of congestion is a simple percentage value, describing the number of tasks in the entire system model part of any chain. Where one hundred percent would indicate that every task is part of one or more chains. These two representations of congestion are multiplied together to produce one value that represents the total congestion. Further it is important to notice that the value of the congestion is not linear, it will vary somewhat based on many factors, such as task amount, chain amount, utilization and so forth. Although, it will provide a general indication how the result will decrease while congestion increases. This is the purpose of the congestion value, not to provide a one-to-one number that can be used to predict the schedulability. It is supposed to be an indication, when a system could consist of to many chains per task.

**Algorithm 5** Calculation of the value representing total congestion in a given system model.

```
1: procedure GetCongestionValue
2:     $taskInChainCounter \leftarrow 0$
3:     $overlapCounter \leftarrow 0$
4:     for $\forall \tau_i \in \Gamma$ do
5:         $localCounter \leftarrow 0$
6:         for $\forall \zeta$ do
7:             if $\tau_i \in \zeta$ then
8:                 $localCounter \leftarrow localCounter + 1$
9:         $overlapCounter \leftarrow overlapCounter + localCounter$
10:        if $localCounter \not 0$ then
11:            $taskInChainCounter \leftarrow taskInChainCounter + 1$
12:    $overLap \leftarrow overlapCounter / taskInChainCounter$
13:    $tasksInChain \leftarrow taskInChainCounter / \Gamma$
14:    return $overLap * tasksInChain$
```

Result of the experiment provide an indication that the success rate in regards to end-to-end timing requirements, is also bound to the task congestion in the system. Additional tasks present in the system seems to not necessarily decrease the schedulability. Further, meeting end-to-end timing requirements also seems easier when additional tasks are present in the system, in some cases. This solidifies the importance of the relationship between the number of tasks in the system and the number of wanted chains. The results show that while this is taken into consideration, the result of generating schedules fulfilling all end-to-end requirements are good in terms of success rate. Especially when comparing to scheduling without the use of job-level dependencies. Another

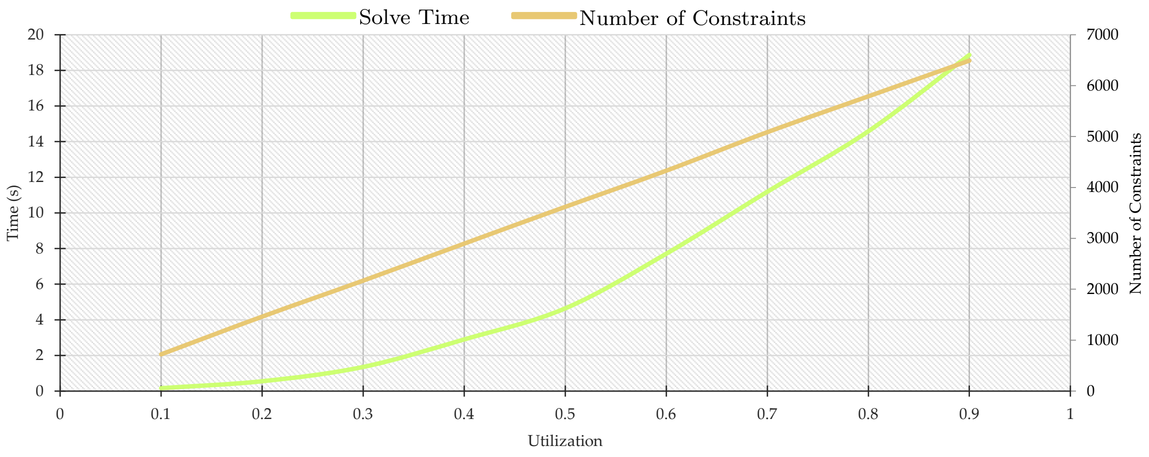


Figure 22: Showing solve times for a system model without any chains. While the utilization increases the number of tasks also increase, as are the number of constraints. While the constraints increase in a linear manner the solve times exponentially increase.

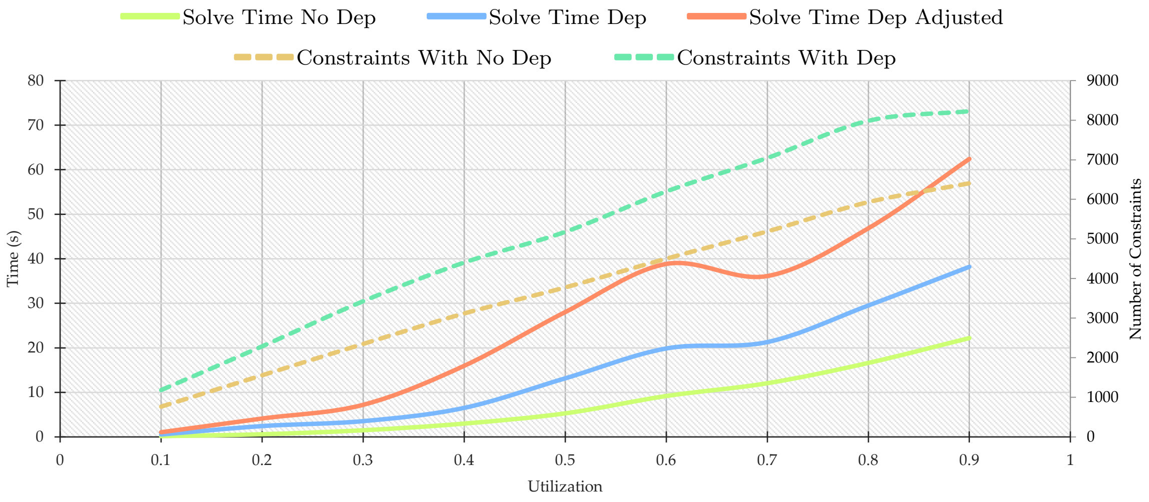
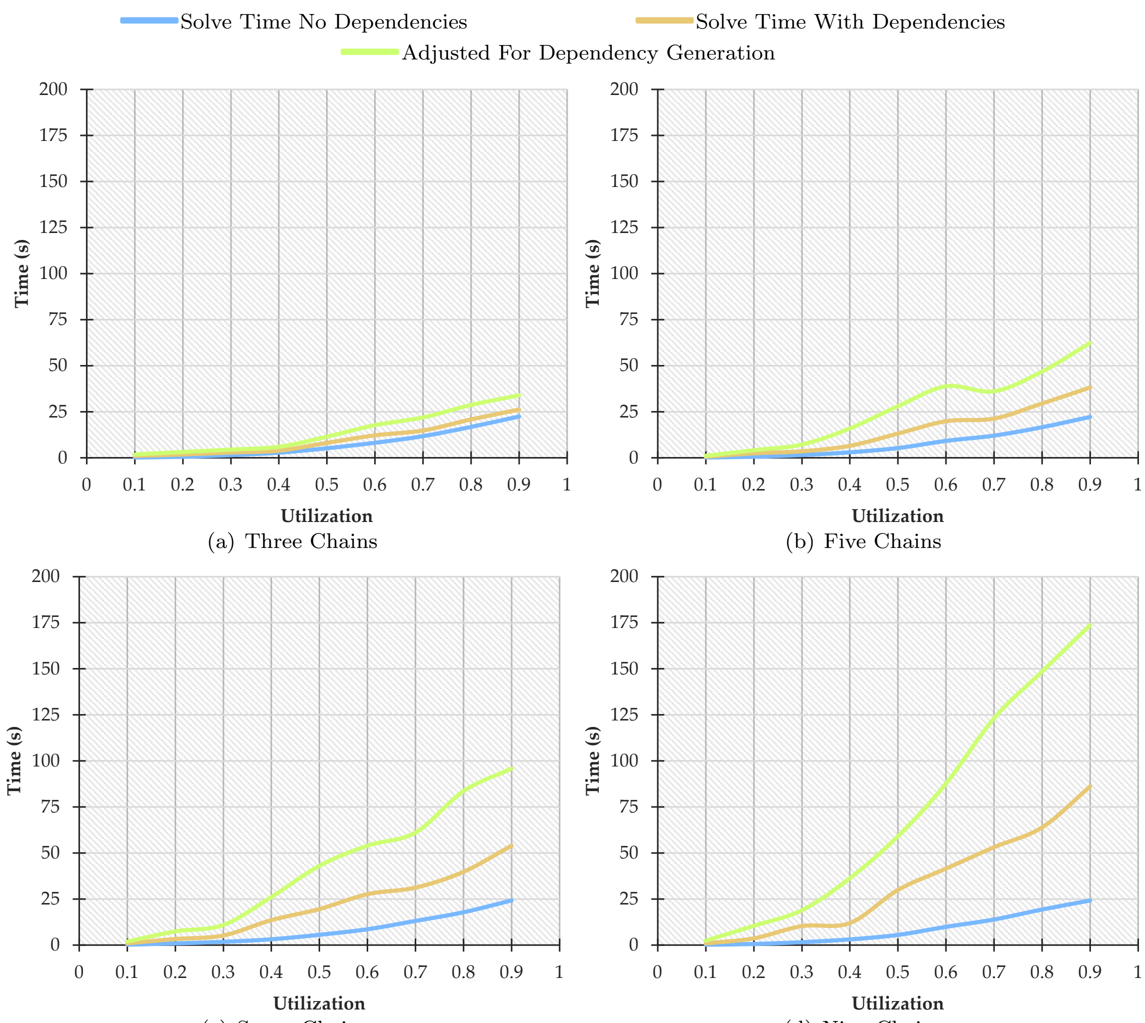


Figure 23: Showing solve times for a system model where five different chains are present. Additional chains increase the number of constraints needed, in turn increasing the solve times. Additionally when accounting for job-dependency generation time the total amount increases even further.

aspect to observe, is that the success seem to be mostly bound to the task-chain congestion more so than the utilization of the system. Considering that the performance seems to stay the same regardless of the system utilization. This is very good since, the schedulability is mostly bound to how the system is characterized, the number of tasks and chains, but the utilization itself seems to have a non-negative impact. Nevertheless the major impact of this approach is the calculation times, that increase the larger the system is and the more constrains that needs to be created for the CP solving.

## 7.3 Generation and Solve Times

Another important aspect that needs to be analysed is the performance of the schedule generation in regards to solve times. All tests have been performed on a laptop with a Intel Core i7-4710HQ 2.50 GHz CPU. Knowing how well the method performs in regards to time could be vital to understand the effectiveness of the approach. Considering the time it takes to generate a schedule, could impact the industrial applicability of the approach. Understandably, as the size of the system models increase the solve times will increase along with it. The reason solve times increase when more tasks are part of a system model is that the number of CP constraints needs to be increased



(a) Three Chains (b) Five Chains

(c) Seven Chains (d) Nine Chains

Figure 24: Shows how solve times evolve when more chains are added.

to schedule the additional tasks. Each individual task instance in the system equals an additional CP constraint. Figure 22 based on 500 iterations, represents the average solve times for a system model without any task chains present. With the purpose to give an indication of solve times, when simply the number of tasks increase in the system along with the increase in utilization. For the test represented by the figure, system models are generated from ten to ninety percent and solve times are measured for each point of utilization. It can be seen that the number of needed CP constraints necessary to define the scheduling problem is increasing linearly, while the solve times increase in an exponential manner.

Therefore, an assumption can be made that an increase in constraints will have a potential impact on the final solve times. This can be seen in Figure 24, where three, five, seven and nine chains have been added to the system model, in four different experiments. Usage of job-level dependencies, to meet end-to-end delay defined by the chains, imply additional CP constraints are needed to fully define the scheduling problem. This increases the complexity of the CSP resulting in larger solve times as can be observed in the graphs. This part also relates to research question 1(b). In addition the increase in solve time for the CP problem, additional time is also spent generating job-level dependencies. Therefore if this additional time is taken into account the schedule generation time is increase slightly more. As can be seen the time stays essentially the same when solving without dependencies, since the size of the systems are still generally the same. Although accounting for the chains causes solve times to increase. Further, the number increases even more when additional chains are used, especially the generation time for the job-

| Name | ID | Period and Deadline | WCET |
|---|---|---|---|
| Task 1 | 1 | 200 ms | 25 ms |
| Task 2 | 2 | 1000 ms | 75 ms |
| Task 3 | 3 | 100 ms | 25 ms |
| Task 4 | 4 | 500 ms | 50 ms |
| Task 5 | 5 | 100 ms | 25 ms |
| Task 6 | 6 | 500 ms | 50 ms |

Table 5: Task set used to perform the complete performance evaluation.

| Name | ID | Tasks in Chain | Maximum Age |
|---|---|---|---|
| Chain 1 | 1 | Task 1 → Task 3 → Task 5 | 225 ms |

Table 6: Task chains used to perform the complete performance evaluation.

level dependencies increases.

Increase in scheduling times when using job-level dependencies are seemingly very dependant on the size of the chains and the total number of chains in any given system model. In Figure 23 solve times in relation to the number of generated constraints, when including five chains in a system model, is displayed. It is evident that additional chains increases the number of constraints necessary, that in turn will result in greater solve times. Also the number of constraints is increasing even more for every extra chain added. It can also be noted that the solve times do not increase in the same ratio as the constraint amount. Hence, this further solidifies the exponential increase in solve times, compared to the number of constraints. Furthermore, it is important to note that it is not really enough to simply look at the CP solve times when utilizing job-level dependencies. When accounting for the generation times of the job-level dependencies the total solve time sometimes doubles. Meaning that for those incidents generation of the dependencies requires essentially the same amount of time as the actual CP solving. This entails that the solve times increase even more rapidly once the number of chains in the system goes up.

## 7.4 Complete Process

The following section aims to provide an overview of the entire process, going from a system model to a valid schedule. This includes starting from a set of tasks and chain, generating a schedule fulfilling all timing requirements and finally executing the provided schedule on real hardware. With the main reason to showcase the process as a proof-of-concept more so than any sort of hardware execution benchmark. Meaning that a simpler task model will be used to generate the schedule, with the intention to make the process more observable and understandable. Technically, additional evaluation on hardware could be performed although it will not provide any additional informative data, considering the focus of this thesis is the schedule generation and not execution. Nevertheless, this experiment shows that a generated schedule will be executable and not just a theoretical schedule.

As stated, the purpose of the experiment is to provide a overview of the process. Hence, the base model of the system is simplified, compared to a potential real system. For the test six different tasks comprise the system, they can be viewed in Table 5, providing a total utilization of 90 percent. Although this utilization is very high and most likely would not be the situation in a real system, it makes it easier to showcase the functionality. Further, the system also contains one chain with its definition displayed in Table 6. It is evident that the system is non complex in nature, although this system has been designed in such a manner that it cannot be validly scheduled without the usage of job level dependencies. This has been done to showcase how the dependencies can change the layout of the schedule, with the purpose to fulfil all end-to-end timing requirements.

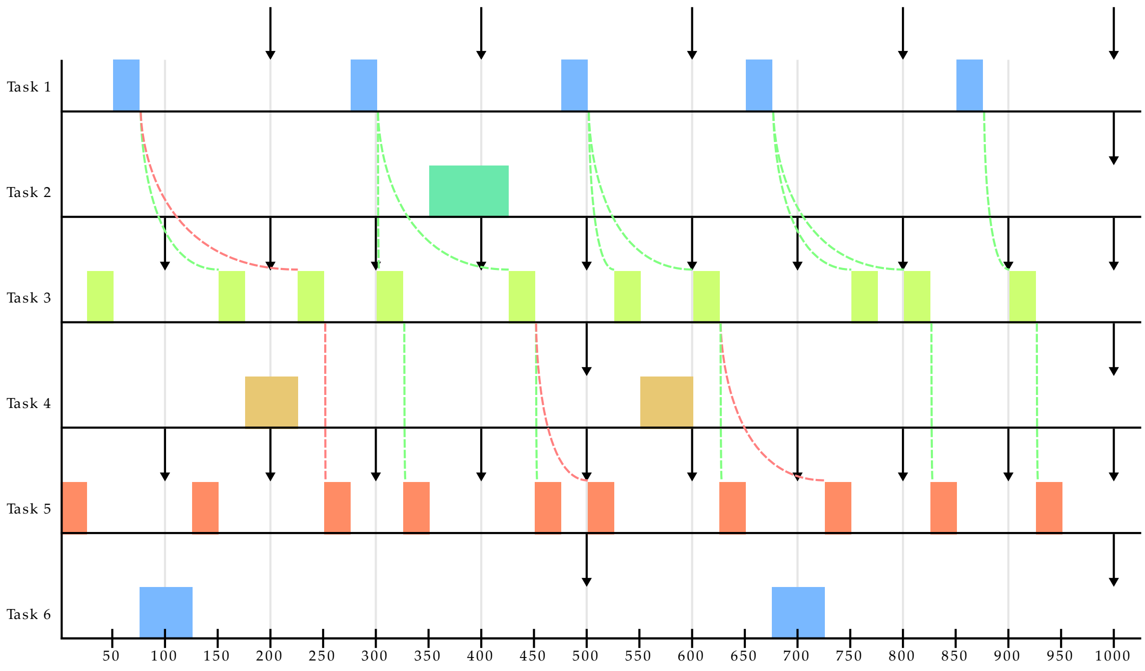


Figure 25: A trace showing a generated schedule for a simple model using no job-level dependencies. It can be seen that not all end-to-end timing requirements are fulfilled, meaning this schedule is not valid.

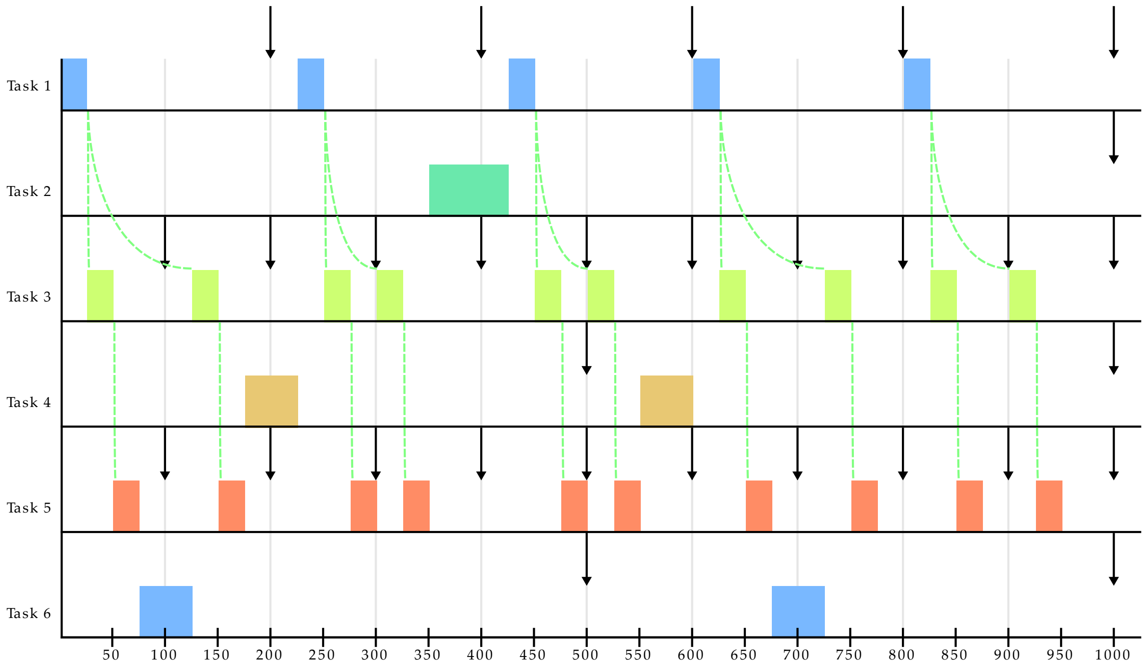


Figure 26: A trace showing a generated schedule for a simple model using job-level dependencies. It can be seen that all end-to-end timing requirements are fulfilled, meaning this schedule is valid.

When the system model has been defined, the next step is to generate a schedule. In Figure 25, a trace can be observed of the schedule of the system without the usage of job-level dependencies.

As can be seen, the schedule is valid in regards to individual task periods, although the end-to-end timing requirements defined by the Chain 1 is not successfully met, as can be seen by the red dashed lines. Hence, this schedule would not be considered valid. Ensuring the end-to-end timing requirements specified by the maximum data age constraints, requires accounting for job-level dependencies when defining the CP problem. For the system considered in this test, the following job-level dependencies are generated: $Task3 \xrightarrow{(1,1)} Task5$ and $Task1 \xrightarrow{(1,1)} Task3$. Subsequently, generating a schedule with the dependencies in mind produces a slightly different schedule from before, a trace can be seen in Figure 26. In contrast to the previous trace, this meets both period deadlines as well as all the end to end timing requirements defined by the chain, shown by the dashed green lines.

Once a valid schedule has been obtained it will be translated into a dispatch table, containing the start and run-times for all tasks in the system. This can be used to execute the provided schedule on a hardware platform. As mentioned earlier an Atmel AVR32UC3A was used as a test platform. When executing the schedule all the start and end times for each task instance execution was logged, with the numbers viewable in Appendix C. When comparing the execution numbers to the trace in Figure 26, the tasks are released and executed within the intended time periods. This entails that the schedule could be successfully executed. Although the system model in regards to the number of task and chains is fairly small, it still provides a proof-of-concept that the entire process itself from generation to execution works. Hence, this provides an answer to the research question 2. That the scheduling on real hardware execute within the desired intervals.

# 8 Discussion

The work performed for this thesis present some interesting discussion points. These can be divided into a few different main topics that assess different parts of the presented scheduling approach. The discussion is mostly focused on reasoning over the obtained results from the experiments and what causes the presented approach to behave the way it does. Thus, the major points of interest are how the solution performs in regards to meeting end-to-end timing requirements, in addition to the performance in regards to schedule generation times. Additionally some interesting limiting aspects will be assessed.

## 8.1 Scheduling Performance

One thing that should be noted when comparing the improvement when considering job-level dependencies as opposed to when not, is that the improvement is hard to quantify with one value representative of all situations. That is because the improvement is very dependant on the system model to schedule. In regards to the size of the system, number of jobs and tasks, chain amount and so on. In many cases the improvement is more than doubled for system utilizations that could be utilized in a real system. But at very high utilizations the improvement can be extreme, although this is because the schedulability ratio without dependencies considered is almost zero. Considering this is important when attempting to evaluate the improvement. It is very important to consider the system model used and treat the improvements separately in a way.

In regards to meeting the end-to-end deadlines, defined for any present task chain, it can be observed that the schedulability ratio decreases along with the increase in number of present chains. Seemingly the schedulability ratio stays mostly the same when the utilization increases, this would point to that the greatest impact on the success rate would be the relationship between the number of tasks and the number of chains defined for a system model. This relationship represented as task-chain congestion, has been observed to increase as the success rate decreases. Considering if the number of chains increase while the number of tasks stay essentially the same, a greater number of tasks will end up being part of one or more chains. This causes the systems to be increasingly difficult to solve, since some situations may arise that simply can't be associated with any solution. Its important to know that solving for a schedule using CP is optimal. Meaning, that if no solution can be found it does not exist. Therefore this seemingly points towards task-chain congestion as the factor having the largest impact on the success rate meeting end-to-end timing requirements. Hence, it is vital that when system models are designed this is taken into account. If the congestion is very high the likelihood of finding a valid solution decreases.

Additionally, this can also explain why in some of the experiments the result improves when the utilization increases for the same chain amount. This is because more tasks are added to the system to achieve a larger utilization, hence the congestion decreases. Nevertheless it should not be expected that the system will always perform better at larger utilization points. Larger utilization still contributes to increased difficulties in obtaining a valid schedule, since less space to distribute each instance exist. Although, for the experiments performed in this work the biggest impact seemingly arises from the congesting. Additional experimentation would be required to see how the performance of the system evolves when even larger amounts of tasks and chains are used. Performing these experiments requires more time, since the solve times increases by a large number when larger system models are used. Also considering the test requires many iterations to get any accurate result.

Another interesting aspect to consider is how the task chains are represented. In this work the chains are constructed using whole tasks. In contrast to the approach used in [36], where each chain is defined by something called runnables. In turn a number of these runnables are used to represent entire tasks. This is somewhat different from how chains are represented in this work, considering the tasks are not made up of smaller parts. Therefore, in this work the same number of chains cannot be created since they are constructed in a different manner. Although, its possible to create tasks that have a smaller WCET to generate a larger number of tasks that can be more similar to these so called runnables. Consequently, it is important to consider how the chains are represented since it can cause the result to appear different.

## 8.2 Solve Time Performance

Addressing how solve times will evolve as increasingly difficult system models are provided is not trivial. When using job-level dependencies to ensure end-to-end deadlines, the increase in necessary solve time is not only bound to the pure number of extra CP constraints. The intricacy of the chains, part of any given system model, impact how difficult its corresponding constraints will be to fulfil. Because the amount of chains and the number of tasks part of each chain along with their internal relationship in the entire system play a big part. Further the task-chain congestion in the system also impact how difficult the additional CP constraints will be to fulfil. Therefore when considering all of these aspects it becomes evident that predicting solve time performance is very difficult. Because it is not just bound to the number of constraints, but also the specific demand placed on the scheduling by the constraints. Considering this, to keep solve times within a reasonable time frame it is advisable to consider the definition of the system model, especially in regards to its task-chain congestion. Nevertheless, prediction of the solve times will mostly not be a trivial task.

Also an important aspect to account for in regards to generation times for the schedules, is the generation time for the job-level dependencies. When accounting for this the total generation time can increase by a considerable margin. When additional chains are added the increase in job-dependency generation time is also behaving in an exponential fashion. One reason that the increase is so much larger for the dependency generation when more chains are considered, compared to the solve time for the schedule, is the fact that more paths need to be considered. When the dependencies have been generated they are simply applied to the scheduling problem. This naturally increases the amount of possible solutions that needs to be considered, but not the same amount as when creating the dependencies.

## 8.3 Other

Seemingly many aspects are tied to the task-chain congestion. High congestion both have a great impact on the performance in both success rate as well as execution times. Therefore to have the ability to judge the results in a manner that extracts something meaningful, this congestion needs to be accounted for. Consequently, simply looking at the percentage results will not tell the entire story. This also impels that constructed systems in reality need to be very aware on the congestion between the chains and tasks used in any given system model. Since the method utilized during the schedule generation will explore all possible solutions, meaning that if a schedule exist it will be found. Therefore, it is vital when designing a real system to account for the relationship between the tasks and the chains.

## 8.4 Difficulties

One of the major difficulties in this thesis was the automatic generation of the system models, to utilize for testing. It needed to be diverse enough to generate many potential situations while still staying confined within certain parameters. Systems can be generated in many different ways and they can be really different in terms of task properties, chain structure, and so on. Hence, ensuring that the generated test cases could be somewhat representative of a real world system was challenging. Further, because how the generation is performed has a big impact on the results, it was an extra sensitive issue. Considering random generation will never be as well thought through as a real system, it may produce some situations that would never arise in a real system model. Minimizing this as much as possible during the system model generation have been challenging. Nevertheless without random generation of the system models any large scale testing would not have been possible.

Something else that turned out to be a challenging aspect performing this work was the amount of time necessary for evaluation. To produce any form of accurate results many iterations are necessary for any given test. This limits the amount of time that can be spent for each test during the evaluation. Therefore, the evaluation had to be prioritised in regards to quality vs quantity. Since it was necessary to with a bit of both, some of the results may appear to be somewhat jumpy. Although, if time would allow for additional iterations for each test the trends would turn out more smooth. Nevertheless the results should be accurate enough to draw some conclusions.

# 9 Conclusion

In this thesis an approach to generate schedules from a given system model, aimed to fulfil end-to-end timing requirements produced by cause-effect chains, is presented. Ensuring that end-to-end deadlines can be met, so called job-level dependencies are utilized. These present a very useful trait specifying that if the dependencies can be generated for a given system model, any generated schedule taking them into account will always meet end-to-end deadlines. For this work scheduling is attempted using the paradigm of constraint programming. This entails converting the system model to schedule, along with any job-level dependencies into a CSP problem that can be solved in an attempt to produce a schedule meeting end-to-end requirements. The following has been the main focus of this work, how well CP could be applied together with job-level dependencies in regards to schedule generation.

Assessment of the performance has been concluded by observing two major aspects, firstly the schedulability ratio creating valid schedules, secondly performance in regards to generation times. Experiments performed have utilized auto generated system models attempting to resemble automotive system as much as circumstances allow. Generating system models that need to be random enough to represent many possible situations, and still generate systems that exist within the constraints of the domain. When generating a system the size created is dictated by the wanted utilization along with the number of desired chains. Experiments have been conducted by increasing the number of chains for each experiment while observing its behaviour over the entire utilization range. With the goal to observe how the performance develops when the system grows in both task and chain numbers.

Meeting the end-to-end deadlines shows a greatly increased performance when using job-level dependencies part of the schedule generation as opposed to not using them, in many cases more than double the performance. This is to be expected, considering meeting end-to-end deadlines without job-level dependencies is mostly situational and dependant on the definitions of the cause-effect chains. Further, another interesting observation is that performance is mostly the same regardless of the system utilization. This is true for the tests conducted, although this result could potentially change using tasks with smaller WCET resulting in an increased number of tasks, making the system additionally more difficult to schedule. The experimentation also shows that the results fall of when additional chains are added to the system. However the reason for this is that some of the system models generated with a larger number of chains present, is impossible to schedule since the constraints that need to be met increase. Considering this work has focused on created schedules for singe core usage, additionally at this stage no support for preemption is present. When taking into account that the job-level dependencies possesses no information of the scheduling approach along with the type of system it should be used for, dependencies can be created that results in conditions that cannot be met. Because of this the term task-chain congestion have been defined to represent the relationship between the number of chains and the number of tasks in any system model. With increased congestion the schedule is more likely to be unsolvable in regards to the end-to-end requirements. Lower congestion improves the possibilities of a schedulable system, although it is not a liner behaviour. A low congestion does not necessary mean that a system will be schedulable for certain, but it indicates that it is more probable. Therefore it is important to take this aspect into account when formulating a system to be scheduled, otherwise meeting the end-to-end requirements will face increased difficulties. Possible improvements to the current system that could help with increasing performance is discussed thoroughly in the Future Work Section 10.

In regards to schedule generation and solve times, the solution performs fairly well and as one could expect. As the systems increase in size linearly, in the number of needed CP constraints, the solve times increase exponentially. Because every constraint added to the problem increases the number of paths necessary to explore for a solution. Further when chains are added to the systems the solve time naturally increase even more. Translating job-level dependencies into CP constraints necessitates additional constraints for the CSP. Because of this every additional chain added have a fairly big impact on the solve times. Additionally, when the generation of the job-level dependencies are taken into account, the total solve times are increased further. In many cases the generation of the dependencies can occupy the same amount of time as the solving of the CP problem. Nevertheless, even if the solve times in some cases can seem to be large, the obtained schedules always ensure the end-to-end timing requirements. But when a schedule can be obtained that ensure all the end-to-end timing requirements, some additional solve time scan be seen as trivial.

In terms of system development, this approach can be very efficient in investigating if a current system model is executable on the given hardware. Since job-level dependencies will always guarantee the systems schedulability if any valid schedule can be found, while simultaneously fulfilling the requirements of the dependencies. Therefore when a CP problem have been defined using job-level dependencies that problem can be solved for. If no valid solution exist the solver will usually understand that fast. Meaning in many cases the solver will not spend to much time searching for a solution when it does not exist. In addition, if a schedule exist solving performance in regards to time is quite decent. When considering the fact mentioned earlier, that in many cases performing analysis of systems considered in this thesis during development, can be quite difficult. Job-level dependencies made it easier to perform analysis of the system during development Although this is without any knowledge of the underlying system as well as the scheduling approach used. Hence, the method presented in this thesis could also be very usable in development Because this method offers a solution to obtain a valid schedule offline that is guaranteed to be fully schedulable Without the need for platform testing or simulation, since the generation itself has already guaranteed the schedulability of the provided system model. Therefore this approach could be of use during development, because it can provide instant schedulability feedback by simply attempt to generate a schedule. Not requiring hardware deployment or a longer simulation period can simply this process significantly.

Furthermore, it is worth mentioning how well the usage of job-level dependencies can be adapted into a CP problem. Considering CP involves defining various constraints to limit the solution space, it matches well with the job-level dependencies. Since they also represent a precedence constraints so its easy to translate and adapt it into the CP domain. Hence, constructing a schedule that take job-level dependencies into account is very suitable using this approach. The way the approach works and how the schedules are formulated require no additional complicated approach. The important thing when defining the schedule is that the job-level dependency is translated correctly, ensuring the created constraints operate on the correct task instances.

Overall the method present a good approach in terms of performing offline scheduling of the type of systems considered. Answering the originally posed research questions. For question 1, that the provided method can efficiently obtain valid schedules for the types of systems considered. Utilizing the dependencies in the generation provide in many cases more than double increase in terms of schedulability ratio meeting end-to-end timing requirements. When the system utilization is increased the total improvement in terms of schedulability ratio increases even further. Although the results do fall of when additional chains are added it is not because the method fails. The demand of the constraints simply cannot be met at a certain point. The CP approach to scheduling is an optimal solution, hence it will find a solution if it exists. Therefore one major conclusion that can be drawn from this is that the design of the system model is very important, if a valid schedule is to be obtained. Some systems simply cannot be met because of the requirements they impose on the system. But improvements to the current solution can certainly be made as discussed in Section 10, Future work. Lastly the second posed research question is also answered. The produced schedules do perform within the given intervals defined by the schedule Even though the systems executed on the hardware platform was a smaller scale, it still provides proof-of-concept that the generated schedules are executable and work as intended.

# 10 Future Work

The work presented in this thesis could potentially be adapted and improved in some interesting ways, resulting in possible improvements on the results and capabilities of the scheduling approach. This thesis work presents a good foundation that can be improved, considering the current solution can face certain situations that is unschedulable in terms of end-to-end requirements. This is certainly the case when the size of the system models increases. Two obvious major additions that could potentially improve the solution can be observed.

## 10.1 Preemption

Firstly the approach to scheduling in this work is entirely non-preemptive. Meaning that a scheduled task instance cannot be interrupted once started, it needs to fully finish its execution. Adding support for preemption could allow some unsolvable situations to change into solvable ones. Adding support for this behaviour into the CP solution is something that is fully doable. In Figure 27 a comparison between the current solution and a possible solution adding preemption can be seen. The current implementations define constraints between every task instance for each task in the system. Stating that all previous instances of a task needs to finish before the next one is allowed to run. Adding preemption would essentially entail adding additional instance to the existing task instances. As can be seen in the figure a time-slice of 25 time units is used to define how many divisions each task instance needs the have. Afterwards additional CP constraints are defined over the newly created slices of all task instances. This approach would make it possible to schedule smaller parts of each instance in between each other essentially creating an approach that allows preemption.

Nevertheless it is important to note that adding this functionality could potentially increase the processing time required to find a valid schedule by a great amount. Depending on the size of slices used to divide each task instance this increase will vary, but for each time a instance is divided an additional CP constraint needs to be formed. Each additional constraint will contribute to increase solve times, hence increasing the total time needed to produce a schedule. As an example consider a system with 100 percent utilization, 1000 ms hyperperiod, and using a time slice of 100 us to create preemption instances. This would generate an additional 10000 CP constraints, considering the solve time analysis presented in this work this could potentially increase solve times by a significant margin. Therefore it is important to make a decision if the potential in increased success rate meeting end-to-end requirements outweighs the additional processing time that will be added. Since any possible improvement cannot be guaranteed, this would require its own extensive evaluation to provide results confirming or denying any potential performance gain.

## 10.2 Multi-core Adaptation

Serving as possibly the most obvious addition to the current approach would be adoption into the multi-core domain. Considering the state of the industry today, many systems move towards or already utilize a many-core approach. Hence, it is not unreasonable to suggest that the next step for the approach presented in this thesis is a move towards a multi-core adaptation. Currently the approach simply uses a singe core to schedule any given system model. This works well within the bounds that utilization of a single core provides. Nevertheless, at some point the systems start to increase in complexity that places the solution outside the limits of a one core approach. Especially in regards to fulfilling the end-to-end delay, simply meeting the individual period deadlines would potentially still perform well on a singe core. Although, considering this work aims to provide solutions in regards to scheduling systems that should meet any given end-to-end delay, improvements should be focused on that area. Therefore the obvious improvement would be multi-core support.

Utilization of a number of cores greater than one could potentially increase the performance in regards to end-to-end delay. Although, the point presented earlier still stands, to system model to schedule needs to be residentially defined. If a system has large task-chain congestion, multi-core could possibly also face issues. But if we consider systems with reasonable congestion adding more cores could possible improve overall performance.

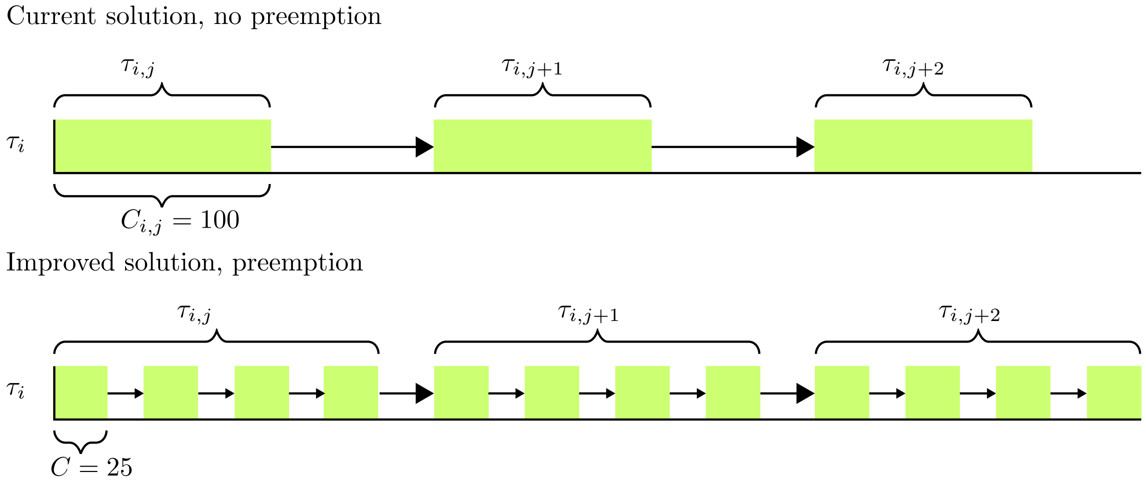


Figure 27: Shows how preemption would change the constraints defined over all the task instances in the system. In this example a task instance have a WCET of 100 and the time slice is 25, meaning that a instance is divided into four smaller parts. These parts behaves in the same way as a task instance and needs to abide by the same rules.

Extending the current CP implementation to support multi-core output, is potentially uncomplicated. When the CSP is defined it is possible to add something called a worker. When solving the defined problem each individual worker could represent a given core. Therefore, defining the cp to solve for multiple cores should be trivial. Nevertheless adapting to multi-core still consist of a major issue that have need to be solved. Division of the individual task instances over the wanted cores. Randomly dividing the task instances over the cores would potentially not increase performance. Therefore some strategy needs to be developed that can distribute all instances over the cores that would result in improvement when solving for a schedule. It is likely this would be a major work within itself, considering the impact the partitioning will have on the final performance on using a multi-core approach.

Regardless of the issues that arise attempting to adapt the system to utilize several cores, it is an interesting area to investigate. If a good approach is found dividing the tasks over the cores, it could potentially increase the performance in terms of meeting end-to-end delays greatly.

## 10.3 Other

Additional minor improvements that could be made to the current implementation include adding support for lower time units. Considering at this time the lowest supported unit is micro seconds. Although, the current limit is enough to represent all the tests that needed to be performed for this work. Nevertheless if a situation would arise that require time units to be used that are lower than micro seconds it would not work in the current state. Therefore this could be an improvment to look into as part of the future work. This also ties into the IBM ILOG CP Optimizer, since the current values supported seems to be limited to 32 bit integer values. This is one of the reasons the lowest time unit had to be changed to micro seconds from nano seconds in the first place. Considering this a, workaround would be required around this limitation.

## 11 Acknowledgements

I would like to extend a big thanks and my utmost gratitude to my supervisor Matthias Becker. A constant source for help and support during the entire thesis, without whom this would not have been possible. I also want to thank my parents for their support during my years of study. Always encouraging and there for me, I hope I can make you proud.

## References

[1] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009.* IEEE Communications Society, 2009.

[2] A. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ecu networks: formal modeling and precise estimation," in *Proceedings of the tenth ACM international conference on Embedded software.* ACM, 2010, pp. 129–138.

[3] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on.* IEEE, 2016, pp. 159–169.

[4] ——, "Mechaniser-a timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies," in *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems WATERS'16, 05 Jul 2016, Toulouse, France*, 2016.

[5] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming.* Elsevier, 2006.

[6] K. Apt, *Principles of constraint programming.* Cambridge University Press, 2003.

[7] I. I. CPLEX, "V12. 1: Users manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[8] P. Laborie and J. Rogerie, "Reasoning with conditional time-intervals." in *FLAIRS conference*, 2008, pp. 555–560.

[9] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "Reasoning with conditional time-intervals. part ii: An algebraical model for resources." in *FLAIRS conference*, 2009, pp. 201–206.

[10] H. Kopetz, *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[11] D. Isovic and G. Fohler, "Handling mixed sets of tasks in combined offline and online scheduled real-time systems," *Real-time systems*, vol. 43, no. 3, pp. 296–325, 2009.

[12] M. Young and L.-C. Shu, *Hybrid online/offline scheduling for hard real-time systems*, 1996.

[13] AUTOSAR, *Specification of Operating System*, Feb. 2009. [Online]. Available: www.autosar.org

[14] G. Fohler, "How different are offline and online scheduling?" in *Proceedings of the 2nd International Real-Time Scheduling Open Problems Seminar*, R. I. Davis and N. Fisher, Eds., July 2011, pp. 5–6.

[15] A. Mohammadi and S. G. Akl, "Scheduling algorithms for real-time systems," *School of Computing Queens University, Tech. Rep*, 2005.

[16] K. Sandstrom, C. Eriksson, and G. Fohler, "Handling interrupts with static scheduling in an automotive vehicle control system," in *Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on.* IEEE, 1998, pp. 158–165.

[17] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multisource software on multicore automotive ecuscombining runnable sequencing with task scheduling," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 10, pp. 3934–3942, 2012.

[18] G. Fohler, T. Lennvall, and G. Buttazzo, “Improved handling of soft aperiodic tasks in offline scheduled real-time systems using total bandwidth server,” in *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on.* IEEE, 2001, pp. 151–157.

[19] “Osek/vdx operating system specification,” OSEK Group, Tech. Rep., 2005.

[20] M. Becker, N. Khalilzad, R. J. Bril, and T. Nolte, “Extended support for limited preemption fixed priority scheduling for osek/autosar-compliant operating systems,” in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2015, pp. 1–11.

[21] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, “Provisioning of predictable embedded software in the vehicle industry: The rubus approach,” in *4th International Workshop on Software Engineering Research and Industry Practice*, May 2017. [Online]. Available: http://www.es.mdh.se/publications/4684-

[22] K. Lakshmanan, D. de Niz, and R. Rajkumar, “Coordinated task scheduling, allocation and synchronization on multiprocessors,” in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE.* IEEE, 2009, pp. 469–478.

[23] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, “Multi-source and multicore automotive ecus-os protection mechanisms and scheduling,” in *Industrial Electronics (ISIE), 2010 IEEE International Symposium on.* IEEE, 2010, pp. 3734–3741.

[24] D. Isovic and G. Fohler, “Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints,” in *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE.* IEEE, 2000, pp. 207–216.

[25] C. Ekelin and J. Jonsson, “Solving embedded system scheduling problems using constraint programming,” 2000.

[26] C. Ekelin, *An optimization framework for scheduling of embedded real-time systems.* Chalmers University of Technology,, 2004.

[27] M. P. Fromherz, “Constraint-based scheduling,” in *American Control Conference, 2001. Proceedings of the 2001*, vol. 4. IEEE, 2001, pp. 3231–3244.

[28] C. Le Pape *et al.*, “Constraint-based scheduling: A tutorial,” 2005.

[29] K. Schild and J. Würtz, “Scheduling of time-triggered real-time systems,” *Constraints*, vol. 5, no. 4, pp. 335–357, 2000.

[30] W. Puffitsch, E. Noulard, and C. Pagetti, “Off-line mapping of multi-rate dependent task sets to many-core platforms,” *Real-Time Systems*, vol. 51, no. 5, pp. 526–565, 2015.

[31] Q. Perret, P. Maurère, É. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, “Mapping hard real-time applications on many-core processors,” in *Proceedings of the 24th International Conference on Real-Time Networks and Systems.* ACM, 2016, pp. 235–244.

[32] J. F. Nunamaker Jr, M. Chen, and T. D. Purdin, “Systems development in information systems research,” *Journal of management information systems*, vol. 7, no. 3, pp. 89–106, 1990.

[33] R. Barry, *FreeRTOS reference manual: API functions and configuration options.* Real Time Engineers Limited, 2009.

[34] *32-Bit Atmel AVR Microcontroller*, Atmel Corporation, 2012. [Online]. Available: www.atmel.com

[35] M. Nasri and B. B. Brandenburg, “Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems.”

[36] S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmarks for free,” in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[37] S. Mubeen, J. Mäki-Turja, and M. Sjödin, “Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study,” *Computer Science and Information Systems*, vol. 10, no. 1, pp. 453–482, 2013.

# Appendix A Example of finalized dispatch table

This section aims to provide an example of the final automatically generated dispatch table. The finalized schedule is exported into the C language so it can be added to an application, and used to build the system during compile time. Therefore, two files are automatically generated from the scheduling software, a h-file and an accompanying c-file. These two files together are enough to provide a fully usable schedule.

DispatchTable.h

```
#ifndef DISPATCHTABLE_
#define DISPATCHTABLE_

#include <stdint.h>
#define totalInstanceCount 30

typedef enum {
        TASK_0,
        TASK_1,
        TASK_2,
        TASK_3,
        TASK_4,
        TASK_5,

        //Counter
        TASK_COUNT
}tasks_IDs_t;

typedef struct {
        uint32_t id;
        uint32_t instanceCount;
        uint32_t wcet;
        uint32_t period;
}tasks_t;

typedef struct {
        tasks_IDs_t parentTask;
        uint16_t currentInstanceCount;
        uint32_t startTime;
}tasksInstance_t;

extern tasks_t SystemTasks[TASK_COUNT];
extern tasksInstance_t sortedTaskInstances[totalInstanceCount];

#endif /* DISPATCHTABLE_ */
```

The h-file provide basic information about how many tasks there are in the system. It also defines two arrays that will contain the actual information about the tasks and the sorted schedule. It is the responsibility of the c-file to define this data.

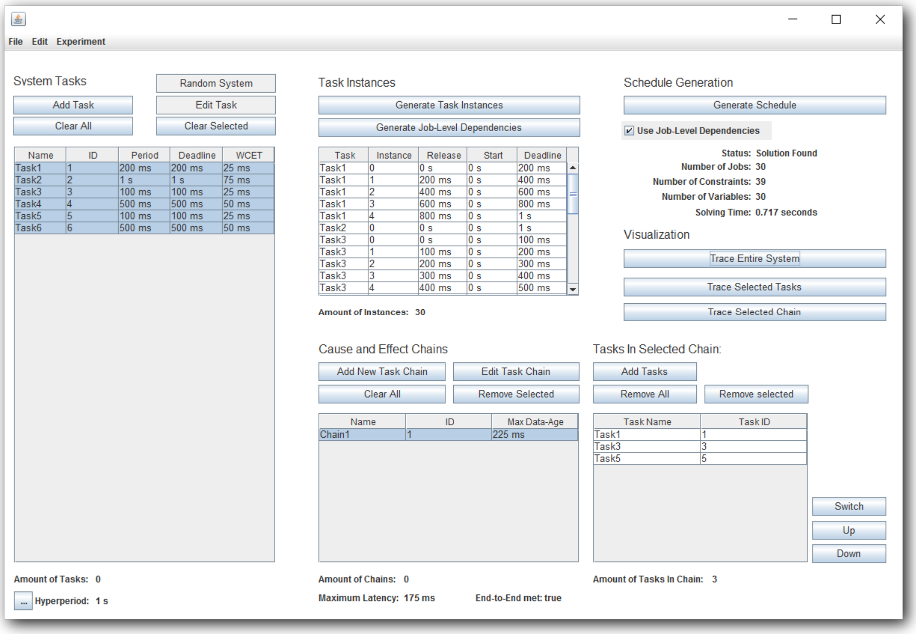
DispatchTable.c

```
#include "DispatchTable.h"

tasks_t SystemTasks[TASK_COUNT] =
{
        [TASK_0] =
        {
        .id = 0,
        .instanceCount = 5,
        .wcet = 25000,
        .period = 200000,
        },

        [TASK_1] =
        {
        .id = 1,
        .instanceCount = 1,
        .wcet = 75000,
        .period = 1000000,
        },

        [TASK_2] =
        {
        .id = 2,
        .instanceCount = 10,
        .wcet = 25000,
        .period = 100000,
        },

        [TASK_3] =
        {
        .id = 3,
        .instanceCount = 2,
        .wcet = 50000,
        .period = 500000,
        },

        [TASK_4] =
        {
        .id = 4,
        .instanceCount = 10,
        .wcet = 25000,
        .period = 100000,
        },

        [TASK_5] =
        {
        .id = 5,
        .instanceCount = 2,
        .wcet = 50000,
        .period = 500000,
        },
};

tasksInstance_t sortedTaskInstances[totalInstanceCount] =
{
[0]  =  {            TASK_0,    0,    0,     },

[1]  =  {            TASK_2,    0,    25000, },

[2]  =  {            TASK_4,    0,    50000, },

[3]  =  {            TASK_5,    0,    75000, },

[4]  =  {            TASK_2,    1,    125000,        },

[5]  =  {            TASK_4,    1,    150000,        },

[6]  =  {            TASK_3,    0,    175000,        },

[7]  =  {            TASK_0,    1,    225000,        },

[8]  =  {            TASK_2,    2,    250000,        },

[9]  =  {            TASK_4,    2,    275000,        },

[10]  = {            TASK_2,    3,    300000,        },

[11]  = {            TASK_4,    3,    325000,        },

[12]  = {            TASK_1,    0,    350000,        },

[13]  = {            TASK_0,    2,    425000,        },

[14]  = {            TASK_2,    4,    450000,        },

[15]  = {            TASK_4,    4,    475000,        },
[16] = {            TASK_2,   5,   500000,        },
[17] = {            TASK_4,   5,   525000,        },
[18] = {            TASK_3,   1,   550000,        },
[19] = {            TASK_0,   3,   600000,        },
[20] = {            TASK_2,   6,   625000,        },
[21] = {            TASK_4,   6,   650000,        },
[22] = {            TASK_5,   1,   675000,        },
[23] = {            TASK_2,   7,   725000,        },
[24] = {            TASK_4,   7,   750000,        },
[25] = {            TASK_0,   4,   800000,        },
[26] = {            TASK_2,   8,   825000,        },
[27] = {            TASK_4,   8,   850000,        },
[28] = {            TASK_2,   9,   900000,        },
[29] = {            TASK_4,   9,   925000,        },
};
```

The c-file provides the information of all the tasks that are present in the system. Also it defines an array of the schedule ordered by task instance start time. It dictates what task should run and what instance of said task, along with the time it should start. When executing the task code itself will execute for the duration of the wcet, provided by the other array defined in the c-file, containing all individual task information. The schedule used as an example in this Appendix, is the same as the one used in Section 7.4.

# Appendix B Prototype scheduling Tool

This section will provide a short description of the tool developed and used to perform the schedule generation experimentation. The interface of the system is very crude and possibly hard to understand. Although since its usage was to provide a simple solution to change the inputs to the schedule generation, not much time was spent on aesthetics. In Figure 28 the main window of the application can be observed. In this window all the tasks and chain that should compose the system are displayed. It is possible to add and change these to suit the system. Further all the generated task instances can also be seen. In the same window the controls for schedule generation can be seen.



System Tasks

| Name | ID | Period | Deadline | WCET |
|---|---|---|---|---|
| Task1 | 1 | 200 ms | 200 ms | 25 ms |
| Task2 | 2 | 1 s | 1 s | 75 ms |
| Task3 | 3 | 100 ms | 100 ms | 25 ms |
| Task4 | 4 | 500 ms | 500 ms | 50 ms |
| Task5 | 5 | 100 ms | 100 ms | 25 ms |
| Task6 | 6 | 500 ms | 500 ms | 50 ms |

Amount of Tasks: 0

Hyperperiod: 1 s

Task Instances

| Task | Instance | Release | Start | Deadline |
|---|---|---|---|---|
| Task1 | 0 | 0 s | 0 s | 200 ms |
| Task1 | 1 | 200 ms | 0 s | 400 ms |
| Task1 | 2 | 400 ms | 0 s | 600 ms |
| Task1 | 3 | 600 ms | 0 s | 800 ms |
| Task1 | 4 | 800 ms | 0 s | 1 s |
| Task2 | 0 | 0 s | 0 s | 1 s |
| Task3 | 0 | 0 s | 0 s | 100 ms |
| Task3 | 1 | 100 ms | 0 s | 200 ms |
| Task3 | 2 | 200 ms | 0 s | 300 ms |
| Task3 | 3 | 300 ms | 0 s | 400 ms |
| Task3 | 4 | 400 ms | 0 s | 500 ms |

Amount of Instances: 30

Schedule Generation

[✓] Use Job-Level Dependencies

Status: Solution Found
Number of Jobs: 30
Number of Constraints: 39
Number of Variables: 30
Solving Time: 0.717 seconds

Visualization

Cause and Effect Chains

| Name | ID | Max Data-Age |
|---|---|---|
| Chain1 | 1 | 225 ms |

Amount of Chains: 0

Maximum Latency: 175 ms End-to-End met: true

Tasks In Selected Chain:

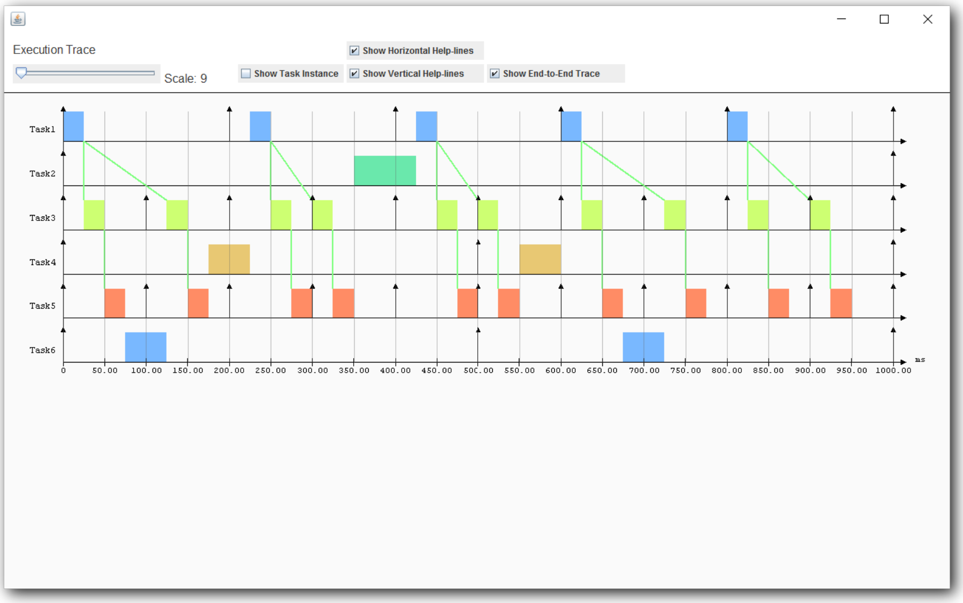
| Task Name | Task ID |
|---|---|
| Task1 | 1 |
| Task3 | 3 |
| Task5 | 5 |

Amount of Tasks In Chain: 3

Figure 28: Showing the main window of the prototype scheduling tool that was developed to perform the work presented in this thesis.

The tool can also be utilized to produce a trace of a created schedule, as can be seen in Figure 29. It is possible to zoom in the trace as well as navigate horizontally and vertically. Further support exist to show helplines as well as showing the instance numbers for every trace block. Another feature is that the trace can visualize the task chain communication, while also showing if the end-to-end timing requirements are met. When visualising the generated schedule it is possible to directly show a trace of the entire system, a selected chain or a individual task. This allow for more careful investigation of a generated schedule.

Additionally a separate part of the tool was constructed to simplify the evaluation process, this part can be seen in Figure 30. In this window automated experiments can be formulated, easing the testing process. During the set-up a system utilization range can be defined, along with the step size and the number of steps. The number of wanted chains in the system model generation can also be assigned. For each of these steps a system model will be generated with the given utilization, and subsequently an attempt to generate a schedule will be performed. Both using job-level dependencies and without. Further, the number of iterations for each point of utilization can also be assigned. This allows the experiment to generate several different system models for each point of utilization. Henceforth, averaged results can be obtained for each given point, providing more accurate results. The tool logs several different parameters, among them: schedulability ratio, solve times, generation times, task set size, and so on. Later the stored parameters can be used during evaluation.



Execution Trace

Scale: 9

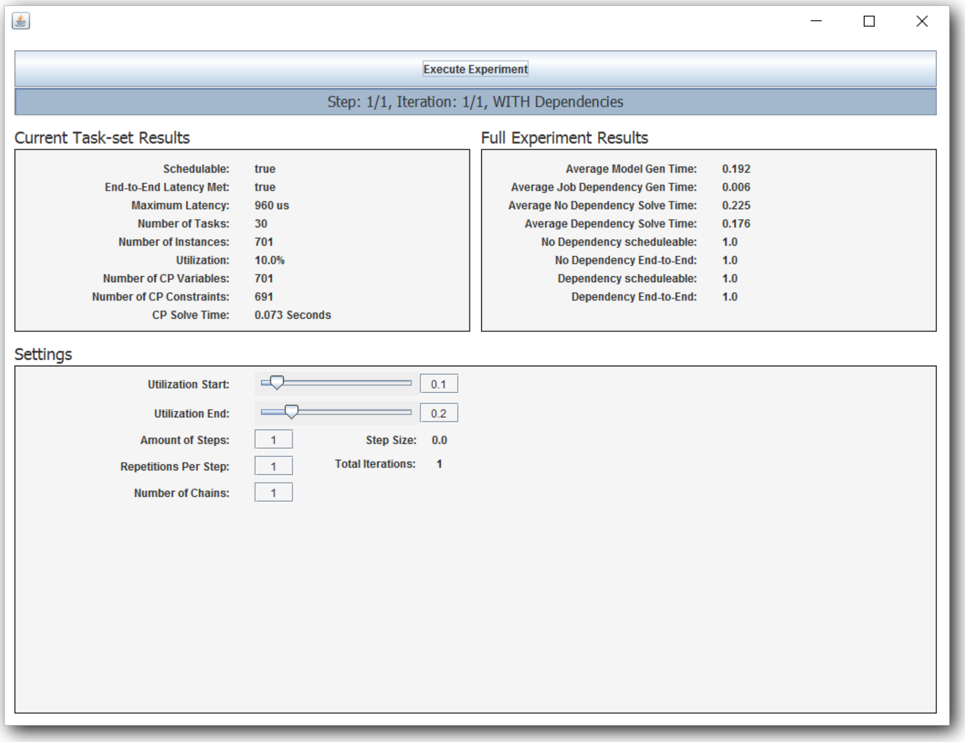
☐ Show Task Instance

☑ Show Horizontal Help-lines

☑ Show Vertical Help-lines

☑ Show End-to-End Trace

Figure 29: Showing the trace window part of the developed tool. This can display any wanted part of the created system schedule.



Step: 1/1, Iteration: 1/1, WITH Dependencies

Current Task-set Results

| | |
|---|---|
| Schedulable: | true |
| End-to-End Latency Met: | true |
| Maximum Latency: | 960 us |
| Number of Tasks: | 30 |
| Number of Instances: | 701 |
| Utilization: | 10.0% |
| Number of CP Variables: | 701 |
| Number of CP Constraints: | 691 |
| CP Solve Time: | 0.073 Seconds |

Full Experiment Results

| | |
|---|---|
| Average Model Gen Time: | 0.192 |
| Average Job Dependency Gen Time: | 0.006 |
| Average No Dependency Solve Time: | 0.225 |
| Average Dependency Solve Time: | 0.176 |
| No Dependency scheduleable: | 1.0 |
| No Dependency End-to-End: | 1.0 |
| Dependency scheduleable: | 1.0 |
| Dependency End-to-End: | 1.0 |

Settings

| | | | |
|---|---|---|---|
| Utilization Start: | 0.1 | | |
| Utilization End: | 0.2 | | |
| Amount of Steps: | 1 | Step Size: | 0.0 |
| Repetitions Per Step: | 1 | Total Iterations: | 1 |
| Number of Chains: | 1 | | |

Figure 30: Showing the window of the tool used to formulate and set-up larger experiments

# Appendix C Hardware platform execution data

| Job | Task | Task Instance | Start | Stop |
|---|---|---|---|---|
| 0 | Task 1 | 0 | 0ms | 25ms |
| 1 | Task 3 | 0 | 25ms | 50ms |
| 2 | Task 5 | 0 | 50ms | 75ms |
| 3 | Task 6 | 0 | 75ms | 125ms |
| 4 | Task 3 | 1 | 125ms | 150ms |
| 5 | Task 5 | 1 | 150ms | 175ms |
| 6 | Task 4 | 0 | 175ms | 225ms |
| 7 | Task 1 | 1 | 225ms | 250ms |
| 8 | Task 3 | 2 | 250ms | 275ms |
| 9 | Task 5 | 2 | 275ms | 300ms |
| 10 | Task 3 | 3 | 300ms | 325ms |
| 11 | Task 5 | 3 | 325ms | 350ms |
| 12 | Task 2 | 0 | 350ms | 425ms |
| 13 | Task 1 | 2 | 425ms | 450ms |
| 14 | Task 3 | 4 | 450ms | 475ms |
| 15 | Task 5 | 4 | 475ms | 500ms |
| 16 | Task 3 | 5 | 500ms | 525ms |
| 17 | Task 5 | 5 | 525ms | 550ms |
| 18 | Task 4 | 1 | 550ms | 600ms |
| 19 | Task 1 | 3 | 600ms | 625ms |
| 20 | Task 3 | 6 | 625ms | 650ms |
| 21 | Task 5 | 6 | 650ms | 675ms |
| 22 | Task 6 | 1 | 675ms | 725ms |
| 23 | Task 3 | 7 | 725ms | 750ms |
| 24 | Task 5 | 7 | 750ms | 775ms |
| 25 | Task 1 | 4 | 800ms | 825ms |
| 26 | Task 3 | 8 | 825ms | 850ms |
| 27 | Task 5 | 8 | 850ms | 875ms |
| 28 | Task 3 | 9 | 900ms | 925ms |
| 29 | Task 5 | 9 | 925ms | 950ms |

Table 7: Showing logged execution data from running a system on the hardware platform.

In this section some execution data is presented that was produced by running a schedule on the hardware platform. The data was logged during execution and written back to the computer using a USART protocol. In Table 7 the data sent back by an execution on the platform can be seen. It shows every individual job in the entire system, its corresponding task, and the job number within that local task. Also the important information is displayed, the start and stop time for the given task instance.