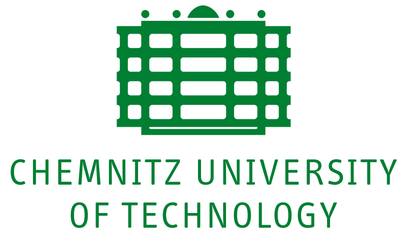



**Faculty of Computer Science**

Chair for Real Time Systems

# Diploma Thesis

## Timing Analysis in Software Development

**Author:** Martin Däumler

**Supervisors:** Jun.-Prof. Dr.-Ing. Robert Baumgartl

Dr.-Ing. Andreas Zagler

**Date of Submission:** March 31, 2008

المنارة للاستشارات

www.manaraa.com

Martin Däumler

**Timing Analysis in Software Development**

Diploma Thesis, Chemnitz University of Technology, 2008

للاستشارات المنارة

www.manaraa.com

## Abstract

Rapid development processes and higher customer requirements lead to increasing integration of software solutions in the automotive industry's products. Today, several electronic control units communicate by bus systems like CAN and provide computation of complex algorithms. This increasingly requires a controlled timing behavior.
The following diploma thesis investigates how the timing analysis tool SymTA/S can be used in the software development process of the ZF Friedrichshafen AG. Within the scope of several scenarios, the benefits of using it, the difficulties in using it and the questions that can not be answered by the timing analysis tool are examined.

المنارة للاستشارات

www.manaraa.com

www.manaraa.com

# Contents


**List of Figures** iv

**List of Tables** vi

**1 Introduction** 1

**2 Execution Time Analysis** 3

2.1 Preface . . . 3
2.2 Dynamic WCET Analysis . . . 4
2.2.1 Methods . . . 4
2.2.2 Problems . . . 4
2.3 Static WCET Analysis . . . 6
2.3.1 Methods . . . 6
2.3.2 Problems . . . 7
2.4 Hybrid WCET Analysis . . . 9
2.5 Survey of Tools: State of the Art . . . 9
2.5.1 aiT . . . 9
2.5.2 Bound-T . . . 11
2.5.3 Chronos . . . 12
2.5.4 MTime . . . 13
2.5.5 Tessy . . . 14
2.5.6 Further Tools . . . 15
2.6 Examination of Methods . . . 16
2.6.1 Software Description . . . 16
2.6.2 Common Measurement Techniques . . . 16
Setup Description . . . 16
INCA . . . 17
Processing Hardware-Traces . . . 18
Summary . . . 19
2.6.3 aiT . . . 19
Preface . . . 19
Analyzing Manually Build Binaries . . . 20
Analyzing Final Binaries . . . 22
Summary . . . 25




المنارة للاستشارات

www.manaraa.com

2.6.4 Tessy . . . 26
Preface . . . 26
Specifying Test Cases . . . 26
Execution Time Measurement . . . 27
Summary . . . 28
2.6.5 Alternative Proposals . . . 29
2.6.6 Result Analysis . . . 31
2.7 Measuring And Processing Execution Times . . . 32
2.8 Summary . . . 33

**3 Scheduling Analysis with SymTA/S 35**
3.1 Preface . . . 35
3.2 Automotive Software Example using OSEK OS . . . 36
3.2.1 OSEK OS Standard . . . 36
3.2.2 OSEK Implementation in Automotive Industry . . . 40
3.3 SymTA/S Internals . . . 41
3.3.1 General Structure . . . 41
3.3.2 Compositional Scheduling Analysis . . . 43
3.3.3 OSEK OS Scheduling . . . 45
3.3.4 CAN Scheduling . . . 46
3.3.5 End-to-End Analysis . . . 49
Preface . . . 49
Event-Triggered Path Analysis . . . 49
Signal-Path Analysis with CAN-communication . . . 49
Signal-Path Analysis with FlexRay-communication . . . 54
3.3.6 Sensitivity Analysis Plug-In . . . 56
3.3.7 Design Space Exploration Plug-In . . . 59
3.4 Scenario 1: Analysis of an Existing System . . . 60
3.4.1 Preface . . . 60
3.4.2 Analysis using Execution Times derived from Hardware-Traces . 61
Analysis Using Task Execution Times . . . 61
Analysis Using Process Execution Times . . . 63
3.4.3 Analysis using Execution Times measured by INCA . . . 64
Scheduling Analysis . . . 64
Sensitivity Analysis . . . 67
Design Space Exploration . . . 68
3.4.4 Conclusion of Scenario 1 . . . 70
3.5 Scenario 2: Extension of an Existing System . . . 72
3.5.1 Preface . . . 72
3.5.2 System Description . . . 72
3.5.3 Scheduling Analysis . . . 74
3.5.4 Sensitivity Analysis . . . 76



المنارة للاستشارات
www.manaraa.com

3.5.5 Design Space Exploration . . . . . . . . . . . . . . . . . . . . . 78
3.5.6 Conclusion of Scenario 2 . . . . . . . . . . . . . . . . . . . . . . 79
3.6 Scenario 3: Timing Budgeting . . . . . . . . . . . . . . . . . . . . . . . 82
3.6.1 Timing Budgeting . . . . . . . . . . . . . . . . . . . . . . . . 82
3.6.2 Control of the Timing Budget . . . . . . . . . . . . . . . . . . . 84
3.6.3 Conclusion of Scenario 3 . . . . . . . . . . . . . . . . . . . . . . 85
3.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

**4 Real-Time Analysis with chronSim 87**

**5 Further Work 90**

**6 Conclusion 91**

**Glossary 93**

**Bibliography 95**



المنارة للاستشارات
www.manaraa.com

# List of Figures


1.1 Software development: V-Model, from [38] . . . . . . . . . . . . . . . . 1

2.1 The WCET problem, from [53, p. 3] . . . . . . . . . . . . . . . . . . . . 3
2.2 Tools for dynamic execution time-analysis, from [22, p. 3] . . . . . . . . 4
2.3 Core components of a WCET analysis tool. The flow of information is shown by arrows filled in grey. The arrows filled white represent tool-construction input. (from [53, p. 11]) . . . . . . . . . . . . . . . . . . . . 6
2.4 Bound calculation methods, from [53, p. 15] . . . . . . . . . . . . . . . . 8
2.5 aiT's phases of WCET analysis, from [6, p. 2] . . . . . . . . . . . . . . . 10
2.6 Bound-T's inputs and outputs, from [55] . . . . . . . . . . . . . . . . . . 12
2.7 Architecture of the TU-Vienna hybrid timing analysis tool, from [53, p. 27] 13
2.8 Tessy's test application consists of original code and generated code, from [50, p. 8] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.9 aiT's (build 73031) call graph analyzing the manually build binary including sub-function five . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.10 aiT's (build 73031) call graph analyzing the manually build binary excluding sub-function five . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
2.11 aiT's (build 73031) call graph after analyzing the small final binary using local worst-cases (the routines are renamed) . . . . . . . . . . . . . . . . 25
2.12 Example source code and related CFG for ITA, from [61, p. 2] . . . . . . 30

3.1 OSEK OS processing levels, from [34, p. 12] . . . . . . . . . . . . . . . . 37
3.2 OSEK OS extended task state model, from [34, p. 17] . . . . . . . . . . . 37
3.3 OSEK OS scheduler, from [34, p. 20] . . . . . . . . . . . . . . . . . . . . 38
3.4 SymTA/S tool suite, from SymTA/S Main manual v1.3 page 16 . . . . . . 42
3.5 SymTA/S compositional scheduling analysis approach illustration . . . . 43
3.6 SymTA/S event models, from [39, p. 4] . . . . . . . . . . . . . . . . . . . 44
3.7 SymTA/S OSEK-task . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.8 CAN-bus architecture, from [13, p. 1] . . . . . . . . . . . . . . . . . . . . 47
3.9 SymTA/S CAN-frame . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
3.10 SymTA/S offset blind analysis Gantt-chart . . . . . . . . . . . . . . . . . 50
3.11 SymTA/S full offset analysis Gantt-chart . . . . . . . . . . . . . . . . . . 51
3.12 SymTA/S example system with buffered CAN-communication . . . . . . 52
3.13 SymTA/S buffered CAN-communication Gantt-chart extract . . . . . . . 53




www.manaraa.com

3.14 SymTA/S asynchronous FlexRay-communication Gantt-chart . . . . . . . 55
3.15 SymTA/S synchronous FlexRay-communication Gantt-chart . . . . . . . 57
3.16 SymTA/S resource speed sensitivity analysis . . . . . . . . . . . . . . . . 58
3.17 SymTA/S jitter dependency diagram . . . . . . . . . . . . . . . . . . . . 58
3.18 SymTA/S design space exploration loop . . . . . . . . . . . . . . . . . . 59
3.19 SymTA/S scenario 1: system model . . . . . . . . . . . . . . . . . . . . 62
3.20 SymTA/S scenario 1: observed paths using task execution times derived from hardware-traces . . . . . . . . . . . . . . . . . . . . . . . . . . 62
3.21 SymTA/S scenario 1: observed paths using process execution times derived from hardware-traces . . . . . . . . . . . . . . . . . . . . . . . . 64
3.22 SymTA/S scenario 1: observed paths using process execution times measured by INCA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
3.23 SymTA/S scenario 1: Gantt-chart of the 10ms-task's critical instant leading to its WCRT (the white bubbles are the task's processes) . . . . . . . 65
3.24 SymTA/S scenario 1: sensitivity analysis results with WCRT constraint of 8000 $\mu$s for the 10ms-task (from report file *RPF05*) . . . . . . . . . . . . 68
3.25 SymTA/S scenario 2: system model, the surrounded area represents the new functionality to be investigated . . . . . . . . . . . . . . . . . . . . 72
3.26 SymTA/S scenario 2: cycle Gantt-chart extract which does not display all blocking CAN-frames due to readability reasons . . . . . . . . . . . . . . 75
3.27 SymTA/S scenario 2: cycle WCRT using *First Through* semantic (the first WCRT does not take the delay determined by SymTA/S between *frame_8* and the 10ms-task on *OSEK-ECU0* into account) . . . . . . . . . . . . . 76
3.28 SymTA/S scenario 2: sensitivity analysis results for *First Through* and *Maximum Age* semantic . . . . . . . . . . . . . . . . . . . . . . . . . . 77
3.29 SymTA/S scenario 2: processor utilization after assigning the maximum WCET to the 10ms-task on *OSEK-ECU0* that is determined by sensitivity analysis using the *First Through* semantic . . . . . . . . . . . . . . . . . . 77
3.30 SymTA/S scenario 2: end-to-end path Gantt-chart extract from scheduling analysis with optimized configuration *First Through* semantic (blocking CAN-frames are not displayed due to readability reasons) . . . . . . . . . 80
3.31 SymTA/S scenario 2: Gantt-chart extract where the maximum WCET of the 10ms-task on *OSEK-ECU0* is assigned to it while using the optimized system configuration and *First Through* semantic (blocking CAN-frames are not displayed due to readability reasons) . . . . . . . . . . . . . . . . 81
3.32 SymTA/S scenario 2: Gantt-chart extract where a WCET of the 10ms-task on *OSEK-ECU0* is assigned to it that is greater than the maximum possible while using the optimized system configuration and *First Through semantic* (blocking CAN-frames are not displayed due to readability reasons) . . 81

4.1 Example of a chronSim task model with specified execution times and operating system system calls (from chronSim presentation material) . . . 87



المنارة للاستشارات
www.manaraa.com

# List of Tables


2.1 WCETs determined by aiT for TriCore v2.0 build 73031, values in brackets are determined by using the annotation *snippet "sub-function5" is never executed;* (mb = manually build, sb = small binary, bb = big binary) 23

2.2 Summary of the obtained WCETs using different methods respectively tools 31

3.1 SymTA/S scenario 1: WCRTs resulting from design space exploration, values in brackets results from rounded activation offsets (compare with initial system configuration (figure 3.22), $\sum$ WCRT: 1030960.23 $\mu$s) . . . 69

3.2 SymTA/S scenario 2: activation offset optimization results for *OSEK-ECU0* 79




المنارة للاستشارات

www.manaraa.com

# 1 Introduction

Nowadays, one can not imagine modern vehicles without microcontrollers controlling the engine, transmission, safety relevant systems like electronic brake or comfort features like power window lifts. Several Electronic Control Units (ECUs) are networked by several bus system like CAN (Controller Area Network) [14], FlexRay [25] or LIN (Local Interconnect Network) [30]. Especially for safety relevant systems a well known timing behavior is necessary.
AUTOSAR [9] is the result of efforts that are made to develop an open and standardized system architecture for the automotive industry. Due to the lack of modeling timing issues, the TIMMO (Timing Model) project [57] was created in April 2007 in order to investigate methods to handle timing information of embedded real-time systems.
The timing analysis tool SymTA/S by Symta Vision [43] is a scheduling analyzer customized for automotive industry. It provides end-to-end analysis as well as an optimization function and a sensitivity analysis. In order to evaluate SymTA/S for ZF Friedrichshafen AG [64] (ZF in the following), three scenarios will be examined which investigate an efficient way of integrating the timing analysis tool in the software development process (V-Model [59], see figure 1.1) of ZF. The first scenario is the timing analysis of an existing

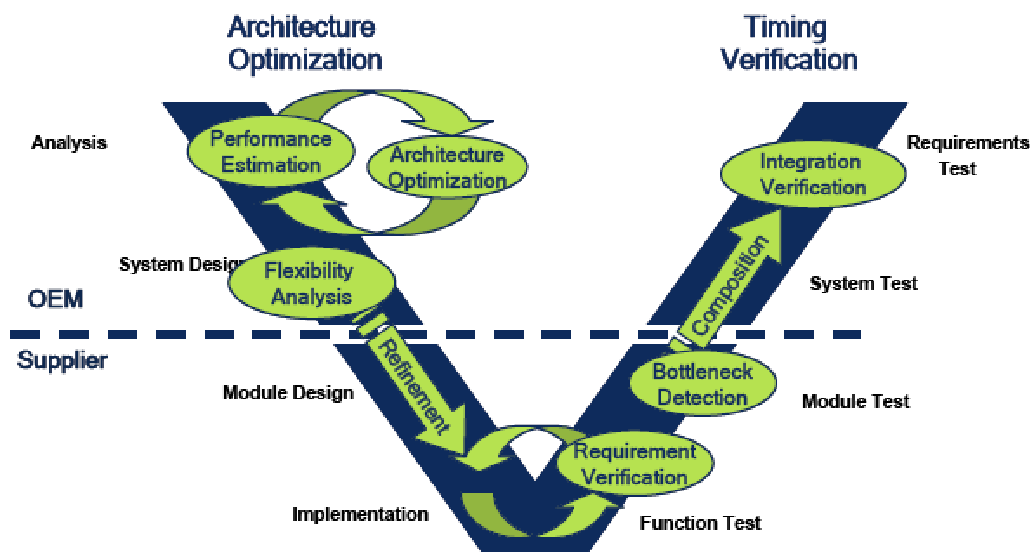



Figure 1.1: Software development: V-Model, from [38]

ECU. Second, the analysis of function expansion on an ECU and the distribution of functions to several ECUs considering the communication between. The last scenario is about



www.manaraa.com

analyzing how the tool can help to give a timing budget early in the development process. For each scenario, it has to be examined how to get the necessary input and put it into the tool as efficient as possible, which improvements in optimization and efficiency the tool provides, when it can be used in development and which problems can not be solved with the tool.
The remainder of this thesis is organized as follows: Chapter 2 describes several methods to gain (worst-case) execution times from code used in software development of ZF. Chapter 3 describes SymTA/S internals and investigates the named use scenarios. Chapter 4 describes another timing analysis tool, chronSim. Chapters 5 and 6 complete this work with an outlook of further work respectively a recapitulating conclusion.



المنارة للاستشارات
www.manaraa.com

# 2 Execution Time Analysis

## 2.1 Preface

A crucial input for timing analysis with a tool like SymTA/S is the worst-case execution time (WCET) of the software that is investigated. The WCET is the longest execution time of code that runs on particular hardware given a particular input. Thus, the input has to be considered as well as the hardware behavior. This chapter should be considered as general overview of several analysis methods and their problems, independent from a specific timing tool fed with this values. [53] gives an extensive summary.
For safety relevant systems, very precise results are necessary in order to guarantee the timing behavior of such a system. So, a given WCET has to be *safe*, i.e. above or equal to the real WCET, and *tight*, i.e. close to the real WCET. Figure 2.1 illustrates the WCET problem. It is desirable to determine the WCET of code efficiently early in development.

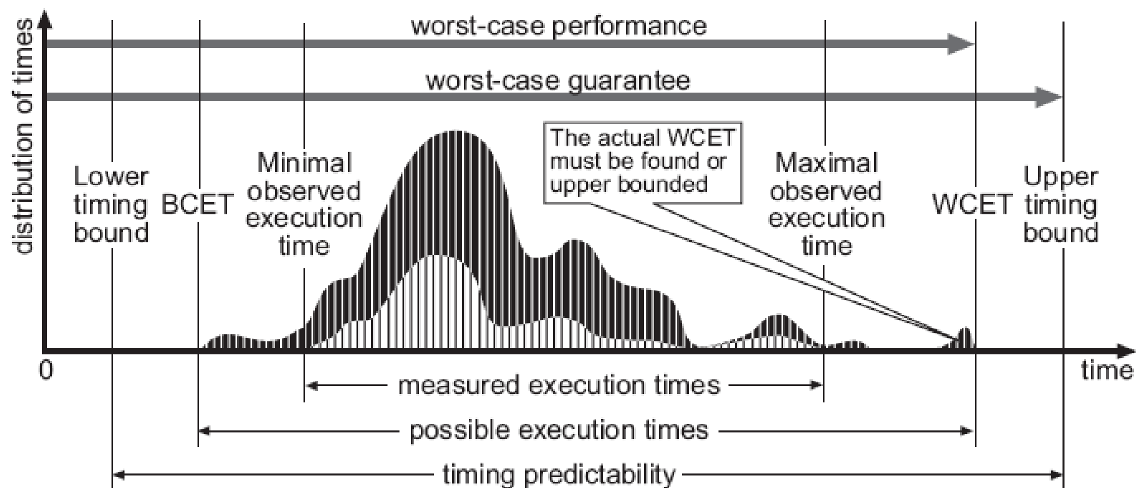



Figure 2.1: The WCET problem, from [53, p. 3]

This would help to detect timing problems as early as possible. There are two WCET analysis approaches: the measurement-based approach called *dynamic* analysis and the *static* analysis.

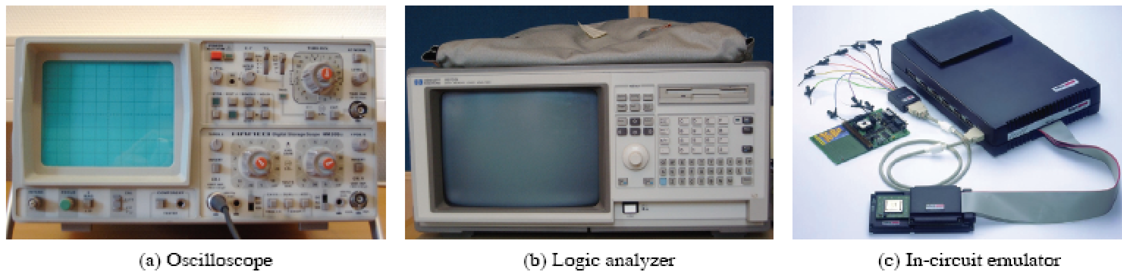


المنارة للاستشارات
www.manaraa.com

## 2.2 Dynamic WCET Analysis

### 2.2.1 Methods

The measurement-based determination of the WCET is widespread in industrial settings [22, p. 2]. The execution time of the binary code on the target is measured for several inputs. In general, each input leads to the execution of one certain path through the code. The execution time of this path is measured.
There are several measurement methods which can be divided into hardware-based and software-based ones [22, p. 2]. **Hardware-based methods** use amongst others an oscilloscope, logic analyzer or in-circuit emulator, see figure 2.2. [22, p. 5] describes that the



(a) Oscilloscope (b) Logic analyzer (c) In-circuit emulator

Figure 2.2: Tools for dynamic execution time-analysis, from [22, p. 3]

oscilloscope has a labor-intensive setup and the source code has to be instrumented in order to take a certain path through the code. The logical analyzer listens to the address-bus and records a trace of the accessed addresses along with time stamps. Another possibility is to use an in-circuit emulator. It also records traces and provides non-intrusive execution time measuring depending on its features.
**Software-based methods** comprise amongst others timing measurement functions provided by the operating system, a cycle-accurate simulator or the instrumentation of the source code with timing measurement code that runs on the target. A cycle-accurate simulator has the advantage that it could be installed on the software developer's host system and therefor could be used for measurement early in the implementation phase without the need to perform the test on the target hardware.

### 2.2.2 Problems

In order to measure the WCET, the input for the path that leads to the longest execution time has to be known. If the code is very complex, i.e. it is a black-box, this WCET input often is unknown. Measuring the code with all possible inputs usually is infeasible due to the **huge state space** when there are many input parameters with a wide range. Therefor, often only a subset of all possible inputs is measured. So, one risks not measuring the worst-case input. The complexity might be reduced by using a divide-and-conquer strategy similar to unit-testing [21], e.g. measuring each single function separated from the



المنارة للاستشارات
www.manaraa.com

whole system and hence testing a straightforward amount of test inputs. This requires system knowledge and black-box testing seems not to be suitable, see section 2.6. The global WCET has to be calculated sophistically on the basis of the measured local WCETs.
The execution times of the processes are **case-sensitive**, e.g. the worst-case input might exclude the simultaneous occurrence of the WCET of several functions or no function of the software has its WCET in this situation. However, this worst-case input respectively this worst-case situation is unknown in general, e.g. if the software is a black-box. Therefor, using the divide-and-conquer approach and calculating the global WCET by simply adding local ones assumably leads to a safe but pessimistic global WCET.
Additionally, there are **interdependencies of the code on hardware level**. Interdependencies can be caused by features of modern CPUs like multi-level caches with a certain replacement policy, translation look-aside buffer (TLB) entries for memory protection or even virtual memory, pipelines, superscalarity with out-of-order execution, (speculative) branch prediction and maybe even multiple cores. So, the worst initial state of the hardware for each measurement has to be known, otherwise possibly yielding a too optimistic result. However, this worst initial state is case-sensitive, i.e. it depends on the code executed before. Regarding and measuring the system as a whole implicitly takes these issues into account but often brakes down on the complexity which causes more test cases than feasible.
A further problem is that **instrumenting the source code** with measurement code affects the timing behavior of the whole system, e.g. by changing the cache contents and extra execution time. When storing the execution time of each process in a separate variable, the number of measured processes is implicitly limited to the available memory. Unfortunately, memory is short in embedded systems. The more precise the measurement software is, i.e. it filters interruptions, the more overhead it has the more it affects the timing behavior of the system. Besides this, the operating system also might influence the execution time because it also uses the hardware.
Code, e.g. a process or a task, that runs on a system might be interrupted by **higher priority tasks or interrupts**. This duration could be calculated by more complex measurement code that is added to the source code. In embedded systems, this method is not a suitable solution due to the hardware restrictions and disabling interrupts is not always possible. Measuring processes in the running system without complex measurement code possibly implicitly includes interruptions into the measured WCETs. So, interrupt service routines (ISRs) have to be triggered during measurement representing a realistic use case of the system. Another possibility is to perform multiple measurements with smaller process subsets while using complex measurement code. A further one is to separately analyze each function and the ISR code and to calculate the global WCET by local WCETs, according to the divide-and-conquer strategy. Then, typical interrupt occurrence scenarios have to be assumed and taken into account during scheduling analysis, e.g. by SymTA/S. Preemptions and interruptions also cause further overhead, e.g. by changing the branch prediction and pipeline behavior, which has to be considered as well in order to gain accurate values. Measuring the whole system implicitly includes this overhead but often is



المنارة للاستشارات
www.manaraa.com

a unsafe WCET analysis method.
Recapitulatory, it is important to note that **measurement-based WECTs are just estimations**. If a huge state space makes it impossible to measure all possible inputs, it is hard to determine if the worst-case input is measured. Adding a safety margin to the measured execution times is an unsafe solution too because it is unknown how much the WCET is underestimated. This could lead to resource wasting when the safety margin is too big and to timing problems when the margin is too small, see section 2.6.3. If it is the intension to give a guarantee that a system will meet its deadlines under all circumstances, e.g. necessary in safety critical systems, a measurement-based WCET is not suitable as input for a scheduling analysis tool.

## 2.3 Static WCET Analysis

### 2.3.1 Methods

In contrast to dynamic WCET analysis, static WCET analysis does not execute the code under investigation. Instead, the binary and maybe the source code are analyzed statically using mathematical models of the software and hardware [8, p. 2]. The calculated WCET qualifies to be safe if the analysis is correct. An upper bound, not just an estimation, is calculated which should be equal or above the real WCET.
Most static WCET analysis methods are divided into three parts: control-flow analysis, low-level analysis and the WCET calculation. Figure 2.3 illustrates core components of static WCET analysis [8, p. 2]. The **control-flow analysis** determines the feasible ex-

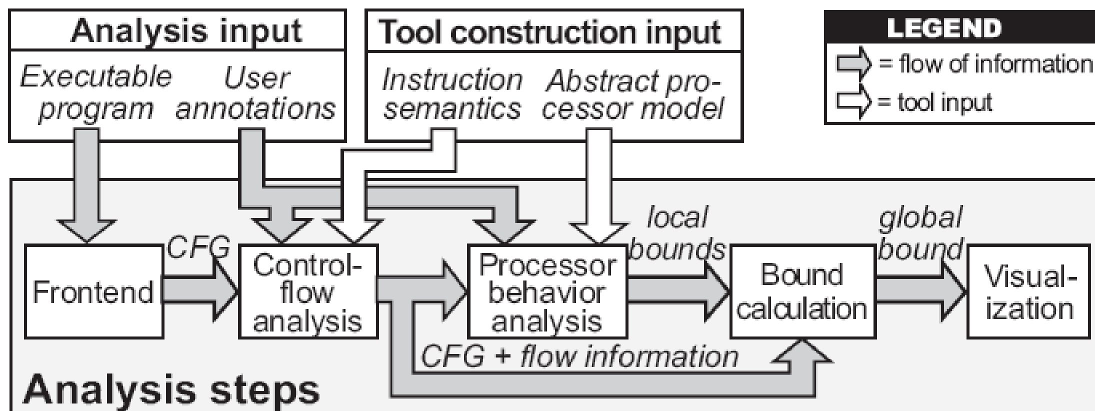



Figure 2.3: Core components of a WCET analysis tool. The flow of information is shown by arrows filled in grey. The arrows filled white represent tool-construction input. (from [53, p. 11])

ecution paths of the examined code. Called functions, loop bounds and dependencies between executable paths are analyzed and infeasible paths are determined that do not



المنارة للاستشارات
www.manaraa.com

have to be considered later. For example, the longest imaginable path through the code would never be taken because of input data dependent conditions on this path. Inputs for the control flow analysis are the call graph and the control-flow graph (CFG), i.e. a representation of the code to be analyzed. These graphs generally are determined from the source code or the assembly code by a so-called *Frontend* of the analysis tool and contain all paths through the code. Additional inputs are input parameter ranges or number of loop iterations and recursions. This information can be provided by user annotations or by a preceding value analysis which is intended to determine effective addresses statically. Effective addresses are necessary for later data cache analysis because data can only be assumed to be in cache if it can be guaranteed. Such a value analysis is implemented in some static WCET analysis tools, see section 2.5.
The **low-level analysis** or processor behavior analysis determines the execution time of the code's instructions on the specific hardware. It has to be taken into account that the execution time of an instruction depends on the instructions executed before respectively on the processor's state. So, the paths that lead to this instruction and to a certain state of the processor and its periphery have to be analyzed. If information is missing, a conservative assumption has to be taken. This low-level analysis has to be aware of modern processor features like caches, pipelines, branch prediction, TLB, etc.
The **bound-, respectively WCET-calculation** determines an upper bound of the global execution time, depending on the results of the path and low-level analysis. This bound should be tight and so a sophisticated approach is needed in order to avoid a too pessimistic global WCET. There are several methods for this calculation, e.g. the path-based and structure-based method as well as an implicit path enumeration technique (IPET). Figure 2.4 illustrates these methods. The structure-based method performs a bottom-up traversal of the syntax tree, combines several statements to a node and determines a new execution time for it by using given rules. These nodes are collected to bigger nodes and so on. The path-based method calculates the execution time for possible paths and returns the longest execution time. However, if there are a lot of branches in the code, the number of possible paths grows very fast (exponentially). IPET uses mathematical constraints to calculate the WCET. In combination with the CFG and the WCETs for each CFG block, the global WCET is calculated by constraint programming or integer linear programming (ILP) with the constraint to maximize the execution time [53, p. 14 cpp.]. Theoretically, this approach is capable to determine the best-case execution time (BCET) as well.

### 2.3.2 Problems

The control-flow analysis is a difficult issue because the **determination of loop bounds, dependencies between paths** and which functions are called is computationally intractable in general. Thus, an approximation might be used [1, p. 4]. So, long infeasible paths might be not detected and are taken into account when calculating the WCET although they are infeasible. Further difficulties arise with dynamic calculated branch target addresses [53, p. 4]. In order to gain a tight WCET, additional user annotations,



المنارة للاستشارات
www.manaraa.com

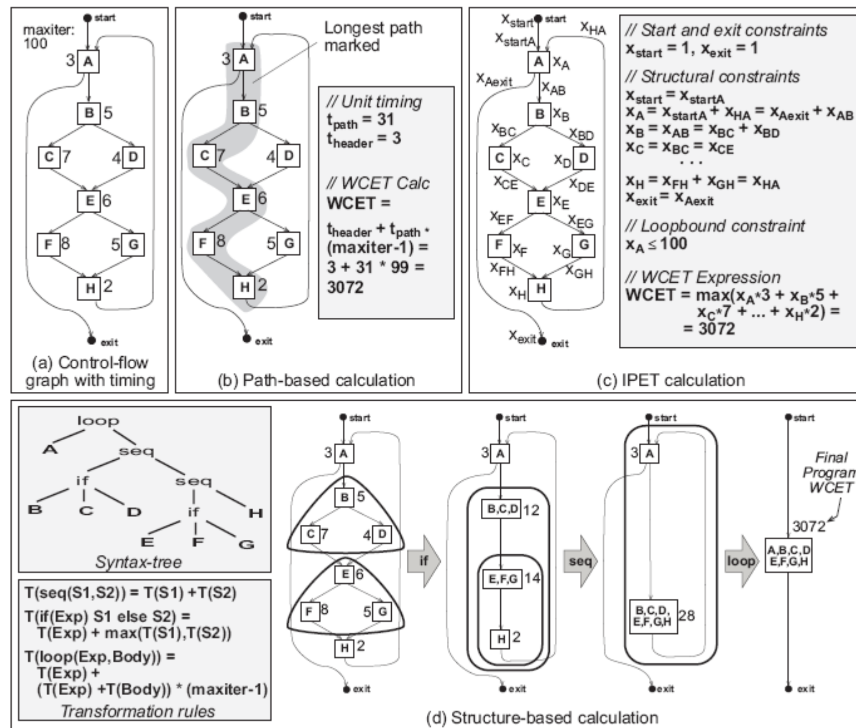



Figure 2.4: Bound calculation methods, from [53, p. 15]

e.g. information about loop bounds, memory layouts, input parameter ranges or infeasible paths, often are necessary. Otherwise, a pessimistic WCET could result [22, p. 2]. This requires deep system knowledge and black-box testing seems to be unsuitable. Section 2.6 comprises results from case studies and own experiences about this issue. Compiler optimizations are a further control-flow analysis related problem. They might change the control flow of the code in comparison to the source code [1, p. 5]. That would make the analysis of the binary necessary.
The correctness and tightness of a statically determined WCET depends on the **underlying hardware model**. Main difficulties are the complex features of modern hardware (see section 2.2.2) which might cause timing anomalies, e.g. a local worst initial state leads to a shorter global execution time, like described in [53, p. 6]. Accordingly, assuming only local worst-cases could be unsafe too. An underlying processor model has to take this into account. That might be difficult if the manufacturer keeps back detailed information about the internals of its processor. Due to the variety of microcontrollers, possibly only those with a high demand on market possibly would be supported quickly by static WCET tool vendors. More information about supported hardware and additional requirements of several analysis tools, like a certain compiler, are described in section 2.5.
The calculated WCET of a piece of code is the execution time without **interrupts or preemptions by higher priority tasks**. They have to be taken into account because they



للاستشارات المنارة
www.manaraa.com

cause additional overhead like context switches and changing the cache contents and the pipeline behaviour, like described in section 2.2.2.
Most available static WCET analysis tools (see section 2.5) are not aware of **object oriented programming languages** like C++, Java or Smalltalk. Furthermore, they require certain programming restrictions like ANSI-C. Problems caused by the object oriented approach are dynamic memory allocation and deallocation as well as dynamic binding [63, p. 3]. When allocating memory dynamically, it is not clear how much time this allocation takes. The allocation could be disabled or an upper bound could be defined on the basis of predefined allocation models. This holds for dynamic memory deallocation as well. Further, a background garbage collector as implemented in Java could lead to unpredictable delays and therefore to very pessimistic execution time predictions. Dynamic binding moves the decision which function is executed to run-time. This possibly depends on input values and could lead to a pessimistic result when always assuming the case with the longest execution time.
Recapitulatory, the main challenge of static WCET analysis seems to be the support of modern processor features in order to overestimate the WCET not too much. In principle, pessimistic WCETs could be used in safety relevant systems. However, this probably causes high costs when, for safety reasons, faster hardware is used than necessary.

## 2.4 Hybrid WCET Analysis

Hybrid WCET analysis approaches combine dynamic WCET analysis with features of the static one. In general, the code (source and/or binary) is analyzed statically to extract several blocks to be measured dynamically. The global WCET is determined by calculating the measured execution times using the methods of static WCET analysis. Additionally, input data could be generated by the analysis tool as well, e.g. by evolutionary algorithms. The hybrid approach avoids the need for a complex hardware model. However, it also suffers from unsafe measured WCETs.

## 2.5 Survey of Tools: State of the Art

### 2.5.1 aiT

The commercial aiT tool by AbsInt Angewandte Informatik GmbH [5] provides static WCET analysis for several microcontrollers. The WCET is determined in several phases which are described in [6] and illustrated by figure 2.5. Inputs for the tool are the executable, user annotations, information about the hardware like memory access timings and buses as well as the start address of the code of interest. The executable can be given in ELF (Executable and Linking Format), COFF (Common Object File Format), a.out and several other formats. As output, aiT provides the WCET and visualizes the program flow,



المنارة للاستشارات
www.manaraa.com

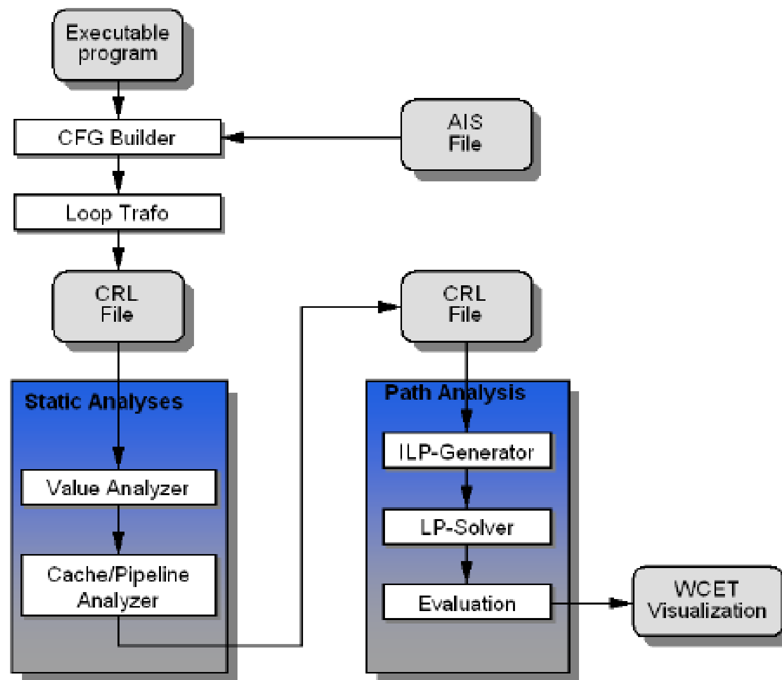



Figure 2.5: aiT's phases of WCET analysis, from [6, p. 2]

call graph, pipeline analysis, see [6, p. 8]. The aiT is used through a graphical user interface (GUI).
The CFG is build by a bottom-up approach [53, p. 20] from the executable. Together with the user annotations it is transformed into a CRL-file (Control-Flow Representation Language), an intermediate format which is input for later analysis phases. After this control-flow analysis phase, the value analysis tries to find the values of registers and variables at a certain program point. If no concrete value can be determined, a safe interval is determined or user annotations are used. So, possible memory accesses and number of loop iterations might be found. The loop bound analysis uses the results of value analysis and a pattern matching method to detect typical loop types. The cache analysis also requires the results of the value analysis in order to classify memory accesses in guaranteed cache hits and potential misses. aiT is able to handle LRU (Last Recently Used), pseudo-LRU and pseudo-Round-Robin replacement strategies [6, p. 4]. It also takes into account that the cache is saturated by the first loop iterations and later ones might have different execution times. These results are input for the pipeline analysis. It determines possible pipeline states for a basic block of the CFG. These possible pipeline states are input for the subsequent basic block whose possible pipeline states are determined instruction by instruction. They are input for the analysis of the subsequent basic block and so on. Output of this analysis is the number of cycles that each basic block needs for execution. Hence, the low-level analysis is finished. The WCET calculation relies on the ILP technique [6,



المنارة للاستشارات
www.manaraa.com

p. 5].
User annotations are necessary if the value analysis fails or in order to refine the results. These annotations can be about calculated branch targets, loop and recursion bounds, infeasible paths, processor clock frequency, the used compiler or the number of separately analyzed loop iterations. Some of this information can be given at source code level by special comments but no further source code modifications are necessary.
As aiT expects an executable as input, it has to be considered how to compile single functions in order to analyze them separately. Otherwise, analysis can be performed first after integrating the whole software which is late in development process. However, compiler optimizations and interdependencies between code snippets might not be considered when analyzing them separately. The used compiler and the programming language (usually ANSI-C) have to be specified. In some cases C++ and Ada are also possible. aiT does not support dynamic programming features and requires the use of the EABI (Embedded Application Binary Interface). Supported targets are Motorola PowerPC MPC 555, 565, and 755, Motorola ColdFire MCF 5307, ARM7 TDMI, HCS12/STAR12, TMS320C33, C166/ST10, Renesas M32C/85 (prototype), and Infineon TriCore 1.3 (from [53, p. 21]) in combination with a specific compiler. A trial version of aiT is available under [4]. Supported operating systems are Windows and Linux. Amongst others, the product family of AbsInt also includes a stack analysis tool (StackAnalyzer), a code compaction tool (aiPop) and a cache analysis tool (aiCache).

### 2.5.2 Bound-T

Bound-T is a commercial WCET analysis tool developed by Tidorum Ltd. Inputs are an executable with debug information (embedded symbol table), the start address of the code of interest and optional user annotations provided by a text file. Supported file formats of the executable are ELF, COFF or AOMF (Absolute Object Module Format) which often dependents on the target processor, see [7] and [10, p. 79]. The outputs are text files containing the WCET and the stack usage as well as files which contain the CFG and the call graph. Figure 2.6 illustrates this. The Bound-T user interface is console based.
Bound-T has an loop analysis in order to determine some loop bounds automatically. [53, p. 22] describes the complex algorithm in detail which is also used to resolve dynamic jumps. Instead of an abstract hardware model, Bound-T uses manually developed processor models based on the processor manuals for the low-level analysis. The WCET calculation bases on IPET using the tool lp_solve [31].
Like aiT, Bound-T needs user annotations in some cases. They are needed when using dynamic calls or dynamic jumps as well as no loop count can be determined automatically.
Bound-T is said to be independent of the programming language. Just compilers with non-standard calling conventions are handled sensitive. Source code annotations are not necessary. Bound-T does not handle code containing recursions. The CFG has to be reducible. [11, p. 14 cpp.] and [53, p. 23] give an overview of the tool's restrictions. It is not mentioned here if Bound-T supports object oriented code. In contrast to aiT, Bound-T



المنارة للاستشارات
www.manaraa.com

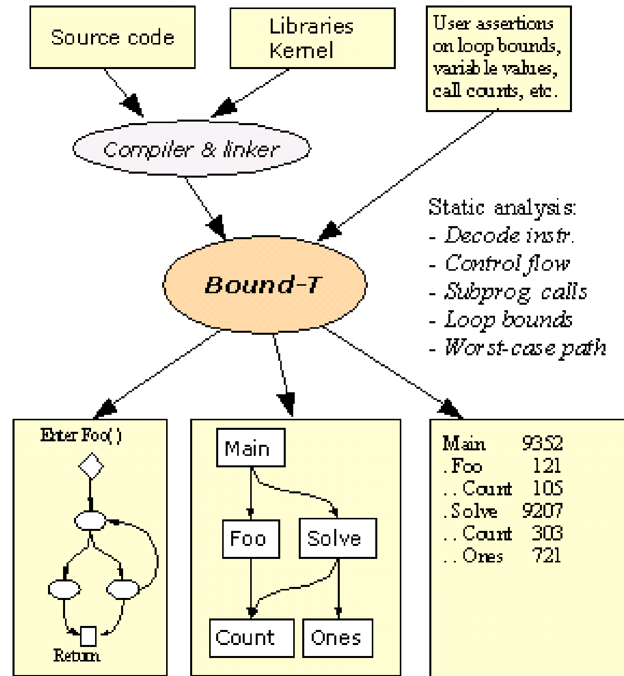



Figure 2.6: Bound-T's inputs and outputs, from [55]

does not perform a cache analysis. Supported processors are Intel-8051 series (MCS-51), Analog Devices ADSP-21020, ATMEL ERC32 (SPARC V7), Renesas H8/3297, ARM7, ATMEL AVR and ATmega (from [53, p. 23]). [24, p. 29 cpp.] describes the porting of Bound-T to the Renesas H8/3297 processor used in Lego Mindstorms. Bound-T supported target platforms are SPARC processor with Solaris operating system and Intel compatible processors running Linux or Windows NT with CygWin. With respect to the use in development process, for Bound-T holds the same as for aiT.

### 2.5.3 Chronos

Chronos is an open source WCET analysis tool developed by the National University of Singapore [16] and is released under GPL (GNU General Public License). Inputs are the C-code file and a description of the target system [17, p. 2 cpp.]. The source code is used by the Frontend for a data-flow analysis in order to determine loop bounds and infeasible paths. Then, a specific compiler is invoked by Chronos and creates the executable from the source code. In the next step, the executable is disassembled in order to build the CFG. After this control-flow analysis, the low-level analysis determines the execution times for each CFG basic block. Chronos is able to analyze out-of-order pipelines, instruction caches and dynamic branch prediction. All results form the WCET of each basic block.



المنارة للاستشارات
www.manaraa.com

The WCET calculation phase uses information about loop bounds and infeasible paths along with the results of the low-level analysis. So, the calculation of the WCET is formulated as an ILP problem. Like Bound-T, it uses the tool lp_solve [31] or the commercial ILP solver CPLEX [28].
User annotations are possible through a GUI or at source code level. They might be necessary for loop bounds and infeasible paths. In order to determine the tightness of the calculated WCET, Chronos provides the simulation of the executable in the SimpleScalar simulator with the same configuration as given for static analysis. This so-called *observed* WCET can be checked against the *examined*, i.e. calculated, WCET, see [17, p. 6 cpp.]. It is not intended to use Chronos for industrial WCET analysis. It provides WCET analysis for the SimpleScalar simulator target, see [52] and [40]. This simulator allows modeling several processor platforms in software [17, p. 4]. The Chronos source code itself, the SimpleScalar simulator, lp_solve and a SimpleScalar-gcc can be downloaded from the Chronons homepage [16]. Chronos does not analyze data cache behavior and the code must not contain recursions [54, p. 9].

### 2.5.4 MTime

The real-time systems group of Vienna University of Technology [62] developed several WCET analysis research projects. There are static WCET analysis prototypes which allow analyzing C-code or Matlab/Simulink models and a measurement-based prototype which generates test data by genetic algorithms. A third project called MTime uses a hybrid approach. It is introduced in the following. More information about the other projects can be found at [53, p. 25 cpp.].
MTime uses techniques of measurement-based and static WCET analysis. Figure 2.7 illustrates its architecture. Input is the C-code. The control flow is analyzed and partitioned

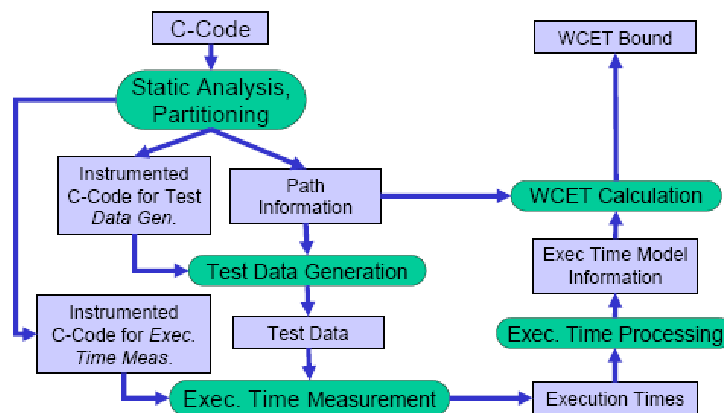



Figure 2.7: Architecture of the TU-Vienna hybrid timing analysis tool, from [53, p. 27]

automatically into segments which are measured later. These segments contain a maximum number of paths which is adjustable. Additional path information which is used to



www.manaraa.com

detect infeasible paths is extracted too. In the next step, test data is generated automatically that should cover all paths in each segment. This is a multiple level process. First, a random search determines the majority of test data inputs. Second, a heuristic like genetic algorithms is used to improve the path coverage. Last, model checking is used to generate the remaining test data. It verifies that a remaining path is infeasible or it generates the test data to take this path. The measurement of the segments is performed on the target hardware. The measured WCETs of each segment along with the additional path information are used for WCET calculation that uses the ILP technique. Measuring the execution time of the segments on the target hardware avoids modeling complex hardware architectures. But it suffers from the problem of dynamic WCET analysis like measurement overhead and source code instrumentation (compare with section 2.2.2). So, MTime does not take different hardware states into account. The calculated WCET can not be guaranteed to be safe. As MTime bases its analysis on C-code, the tool can be ported easily to new processors by changing the instrumentation code. Currently supported targets are the HCS12 and Pentium processors. However, MTime did not support function calls in 2006, see [54, p. 5].

### 2.5.5 Tessy

Tessy is a commercial unit-test tool developed by Hitex. It just finished evaluation phase but is not used in general by ZF. Unit testing stands for extensive and isolated testing of a single unit, for example a single function. Extensively means that the unit is tested with custom made test inputs, for example with unexpected ones. Isolated means that the unit is tested independent from the rest of the application. Therefore, function calls have to be replaced by pseudo-function calls.
Tessy scans C-code and determines the interface of the test object, e.g. the function to be tested. The interface contains the input and output variables of the test object as well as the function calls and global variables. Then, several test cases have to be specified, e.g. one input configuration and the expected output per test case. In case of Tessy, these pseudo-function, so-called stub-functions, just return a test case specific value or source code is provided for them [48]. The tool comes along with a CTE (Classification Tree Editor) in order to define test cases systematically following the classification tree method [58]. More information about systematic construction of test cases can by found at [46].
As unit-tests are on-line, i.e. the test object is executed, its isolation is achieved by the so-called test application approach. A specific test driver is compiled together with the source code of the test object. It contains the startup-code for the specific target, a main function, the call to the test object and possibly code for the stub functions. Figure 2.8 illustrates this. The test application's size primarily depends on the number of input and output variables, not on the number of test cases. Tessy feeds the test application with test case specific input data and picks up the output. The input data is not compiled in the test application.
The test application can be compiled as standalone application. So, it is possible to per-



المنارة للاستشارات
www.manaraa.com

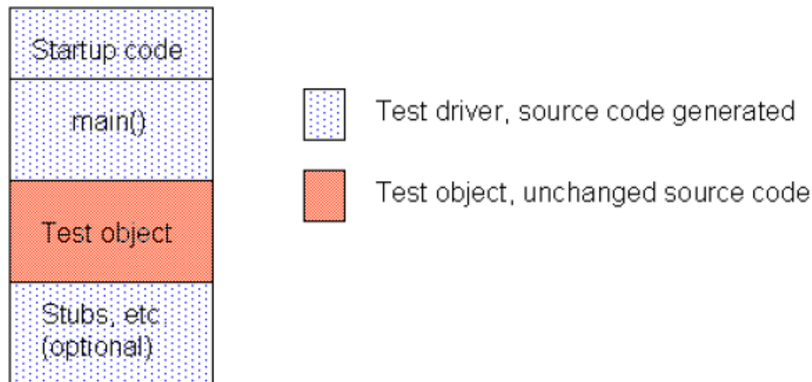



Figure 2.8: Tessy's test application consists of original code and generated code, from [50, p. 8]

form the unit-test on the target. A debugger connected to Tessy acts as interface. Thus, the test application might be run on a simulator, an evaluation board or on an ECU. It is the intention to use Tessy for measuring the WCET of all important paths through certain code snippets, e.g. a function. Then, the global WCET could be determined from those local WCETs (see section 2.6.4). Another test possibility provided by Tessy is the so-called original binary test. The unmodified application as a whole is executed on the target until the test object is reached. Then, the specific variables are set to the test case values and the output is compared with the expected output. This has the advantage that the normal compiler and linker settings can be used. However, the original binary test just can be performed late in development process when an executable software level is build. Furthermore, the use of stub-functions is not possible and the unit is not isolated any more. The test results are provided in common office file formats. Tessy provides line and path coverage analysis of the test cases which can be imported using several file formats. Tessy is also able to work with state machines [49].

### 2.5.6 Further Tools

The foregoing sections give an overview of the state of the art commercial and research WCET analysis tools. There are many more projects whose detailed description would go beyond the scope of this work. OTAWA (Open Tool for Adaptive WCET Analyses) [33] and Heptane [26] are open source WCET analysis tools under GPL. The Chalmers University of Technology developed a WCET analysis research prototype [15]. SWEET (Swedish Execution Time Tool) is a WCET analysis tool prototype which is developed by Mälardalen University [42]. It performs WCET analysis on intermediate code. Therefor, a special compiler is integrated. Further WCET analysis tools are Cinderella [18] and the research project of the Florida State University [53, p. 23 cpp.]. RapiTime by Rapita Systems is a commercial hybrid WCET analysis tool.
In contrast to Tessy which supports only C-code, there are other unit test tools which



المنارة للاستشارات
www.manaraa.com

work on C++ or Java. Commercial examples are Testwell [51] (C, C++, Java), Rational Test Real-Time [37] (Ada, C, C++, Java), VectorCAST [60] (Ada, C, C++) and C++test by Parasoft [12] (C, C++). The software developed by ZF underlies several programming restrictions, like described in section 2.6.1. So, it has to be investigated if a unit test tool like Tessy can handle it. There also exists a number non-commercial unit test tools. [41] summarizes and describes a lot of them. Not all of them allow running the unit-tests on the target.

## 2.6 Examination of Methods

### 2.6.1 Software Description

The following sections give detailed information about methods to gain (worst-case) execution times. This includes the common time measurement technique used by ZF as well as new approaches. As possible, all approaches are evaluated with respect to precision, amount of work and usability in the software development process of ZF. The examination is made exemplary on a piece of automatic transmission software. The extracted example function has a monitoring functionality. It is written in C and is part of a process within a cyclic task with a period of 30 ms. It has two main paths and calls up to five sub-functions. The first path is for error handling and calls sub-function five. In the second path, sub-functions one and two are called to read out a characteristic line respectively map. The result is checked and sub-function five might be called for error handling. At the end of the example function, sub-functions three and four are called in every case. Including all sub-functions, the code counts round 250 lines of code (excluding sub-functions four and five). The example function is simply structured. In general, the code developed by ZF underlies programming restrictions. Thus, recursion, dynamically calculated branch targets and function calls or dynamic memory (de)allocation are currently not used, even if C++ is used. Other functions might contain more branches and more paths than the chosen one. Due to these restrictions, code developed by ZF should be analyzable by common static WCET analysis tools. The corresponding ECU has an Infineon TriCore 1766 32-bit microcontroller for embedded systems with a frequency of 80 MHz running the OSEK OS conform operating system (see section 3.2.1).

### 2.6.2 Common Measurement Techniques

#### Setup Description

The common measurement methods used by ZF allow measuring the execution times of the tasks and/or processes using a specific variant of a final software level, i.e. they are late in development. Section 3.2.2 gives detailed information about the used operating system standard, its configuration and the distinction between tasks and processes. The detailed measurement procedure and precision depend on the actual project. In general,



المنارة للاستشارات
www.manaraa.com

the software is running on the ECU which is connected to the so-called *Laborauto*, a computer that simulates the mechanic of the car, in particular the transmission. The (driver) input for the ECU software is provided by the tool *SoftCar RT*. This tool is developed by ZF. It stimulates the ECU and allows running test scripts which simulate specific driving situations. As smallest software unit, processes can be measured. In order to measure the execution time of the mentioned function, it is moved to an empty process and a new software level is build. One has to keep in mind that this could lead to different execution times as in the original binary due to compiler optimization.

#### INCA

The execution times of the tasks and processes that are running on the ECU are recorded by the tool INCA [20]. INCA reads out the variables of the execution time measurement functionality of the operating system. Depending on the project, the current and maximum execution time of tasks, single processes, ISRs and the processor utilization can be displayed. INCA can record a trace of these values, e.g. when executing a certain SoftCar RT-script. So, this is black-box testing. The results can be processed using common office tools. There is no name-based correlation between INCA measurement variables and the processes to be measured. This selection has to be done manually.
In the example project, interruptions by higher priority tasks are filtered but ISRs not. These ISRs occur more or less non-deterministic. Assume that in a fictitious example, a certain ISR occurs only once within an interval of 100 ms. During the first interval, the ISR interrupts a certain process and lengthens its execution time. During the second interval, the ISR interrupts another process and lengthens its execution time but not the process which it interrupted before. It might be the case that this leads to the maximum measured execution time of both processes. It can not be determined from the measurement trace if this ISR is included twice. Further, it is easily imaginable that two subsequent INCA measurements while running the same SoftCar RT-script provide (slightly) different results. That makes it difficult to assume several ISR occurrence scenarios because the precise number of already included ISRs is unknown. In this case it might be the only possibility to work with ISR-affected execution times. INCA measurement also causes an overhead that might affect the system behavior. Especially when measuring processes of task with a short period, this leads to higher processor utilization.
For the investigated function, a highest execution time of 0.08105 ms is measured while executing a SoftCar RT-script that performs several normal gear shifts and kickdowns. It is intended to cause high processor utilization and interrupt load. It is important to note that it is not guaranteed that this script causes the worst-case situation. Further, the results most probably are affected by interrupts and non-deterministically occurring processes.



المنارة للاستشارات
www.manaraa.com

### Processing Hardware-Traces

Because measurement functions provided by the operating system might affect the system behavior or probably do not take all interruptions into account, a more precise technique using hardware-traces can be used. The software is executed as usual. The output of a hardware debugger which is connected to the ECU via the OCDS2 (On-Chip Debug System) interface is recorded with small (hardware) overhead. Such an output-file (size: round six GiB) is processed by several scripts. These scripts extract the execution times of all tasks, processes and ISRs while considering all possible interruptions. However, there are differences between process and task execution times. In both cases, the interruptions by ISRs and preemptions by higher priority tasks are filtered. Besides the processes which are called deterministically during task execution, calls to functions respectively processes that can not be assigned to a certain task are extracted too. They might be called by the operating system or might stand in conjunction with ISRs. The determined task execution times include them whereas the process execution times do not. So, suitable assumptions would have to be made in order to reproduce the occurrence of these processes, like necessary for ISRs. This causes an unacceptable amount of extra work and is late in software development process. The developer of the scripts adapted them, so that not assigned processes are implicitly included in the process execution times. This makes the results less precise but a separate consideration is assumed to be not manageable.
In principle, the sum of the minimum and maximum process execution times should be less or equal respectively greater or equal than the execution times of the corresponding task after adapting the scripts. SymTA/S report files *RPF00, p. 4* (bases on extracted task execution times) and *RPF01, p. 5* (bases on extracted process execution times) within the enclosed archive summarize the task execution times that base on these different execution times. These report files point out that the sum of the minimum and maximum process execution times cover a smaller range than the task execution times in most cases. According to the developer of the hardware-trace processing scripts, the lesser maximum execution time is caused by an overhead that is considered only at task level. The reason for the greater minimum execution time could not be clarified during this work. The effect of these different results that base on the same measurement on scheduling analysis is described in section 3.4.2.
The processing of hardware-traces is restricted to the OCDS2 interface and also suffers from the unsafety of probably not measuring the WCET. The measuring time of the available debugger is restricted to 850 ms. That makes it hard to measure a specific situation. The analysis time of the scripts take round two hours on the test system. The test system is a 1,4GHz Pentium M machine with 512 MiB of RAM running Windows XP Service Pack 2. This should work better on a computer with a faster CPU. As result, the measured execution time of the process is between 0.0017 ms and 0.0022 ms. This execution time was measured during normal operation, i.e. not while processing a SoftCar RT-script. It is extracted while using the adapted processing scripts that do not filter not assigned processes.



المنارة للاستشارات
www.manaraa.com

#### Summary

Recapitulatory, the common technique bases on the timing measurement functionality of the operating system and INCA. It is unsafe and probably imprecise. It has to be taken into account that the measurement probably affects the system behavior due to the overhead it produces. This overhead and ISRs particularly affects the execution times of small code pieces because hardware-sided interdependencies like a dirtied cache is assumed to have a heavy effect on short execution times. If the execution time of single processes is needed, this method can be time-consuming too. The hardware-trace method provides results that are less affected but also suffer from the unsafeness of dynamic WCET analysis.
The common methods are suitable to get an overview of the average task execution times without great effort. They are independent from the used programming language and could even handle dynamic programming features. So, it might be the only way to get execution times if other methods fail, e.g. for supplied software which is a black-box for ZF. Analysis of the BCET can also be done without great extra effort. But the results are as unsafe and imprecise as the WCET.

### 2.6.3 aiT

#### Preface

The static WCET analysis tool aiT (see section 2.5.1) was examined using a 30-days trial version for Infineon TriCore 1796 and Tasking Compiler [47]. This version includes a web- and telephone-based introduction and support. During evaluation phase, three aiT versions were provided and tested. In particular, aiT for TriCore v2.0 build 63339, build 70825 and build 73031. Problems with the first two version caused consultations of the support team and time-consuming tests. Some WCET analyses took several hours. In the meantime, the test system was not usable. Finally the last aiT version was able to perform a full analysis of the example function.
Refinements of the analysis could be reached through annotations about infeasible paths, (not) executed code, upper loop bounds, stack address, the first CSA (Context Save Area) of the processor and information about the periphery like external memory. For example, the read out of the characteristic map could access a ROM with high access times. In order to reduce analysis-time and resource usage, aiT allows using only local worst-cases. When setting this option, aiT does not follow all successor states of pipeline and cache analysis. Instead, it decides which successor state seems to be the worst and follows it. The resulting execution time might not be the WCET, but it is gained more quickly. This is a good choice when making first analyses in order to determine all necessary annotations. All annotations in order to refine analysis require knowledge about software and hardware.



المنارة للاستشارات
www.manaraa.com

### Analyzing Manually Build Binaries

In order to get a WCET early in development process, an executable that includes the example function has to be build manually from the source code. Using aiT this way might help a software developer to keep a given timing budget by analyzing just written code, see section 3.6. Local WCETs could form the WCET of a function, of a single process and finally of a whole task, according to the divide-and-conquer strategy. Like described in section 2.2.2, this could lead to a pessimistic global WCET. However, if the timing analysis, e.g. by SymTA/S, does not indicate problems based on pessimistic WCETs, this is safe result because the results that are determined by aiT are assumed to be safe.
The manually build binary consists of a main function that just calls the example function and the definition of external variables. Not needed code within the source files is removed. The executable is created by compiling and linking the source files together with a sample locator file which contains a sample memory mapping and with a sample startup-code file. The used compiler is the Tasking TriCore VX v2.2r3p2. If the locator and startup-code files are not specified, the compiler uses default files. For precise analysis, the corresponding files and settings from the current project should be used. Exemplary, creating an executable is done by the following command:

```
~>cctc -g --format=ELF -C tc1766 -d locatorfile.lsl
startup.asm main.c function.c subfunction.c
```

The parameter *-g* causes the compiler to compile in symbolic debug information. This is a useful option for first analyses. If there are problems, the corresponding source code can be displayed by aiT. The parameter *-C tc1766* specifies the target processor. Unfortunately, the aiT version is for the TriCore 1796 whereas the target is a TriCore 1766. However, aiT calculates the same WCETs for executables build with option *-C tc1766* and *-C tc1796*. If not all source code is available, like for the sub-functions four and five in the example, the assumed execution times of them are annotated by:

```
snippet "sub_function4" is not analyzed and takes 39 cycles
and does not violate calling conventions;

snippet "sub_function5" is not analyzed and takes 1718 cycles
and does not violate calling conventions;
```

These annotated WCETs are determined by the analysis of a final software-level binary which is described in the following sub-section. aiT does not simply add these annotated processor cycles. Rather, it takes a certain overhead into account, i.e. pipeline and cache effects, caused by the not-analyzed sub-functions. So, the analyzer flushes the pipeline and fills it after the "call" to the annotated function. When using the annotation appendix *"and does not violate calling conventions"*, aiT assumes that not all value information collected by aiT are destroyed by the not-analyzed code snippet. Instead, it is assumed that the saved registers keep their values and that the user stack remains the same whereas the systems



المنارة للاستشارات
www.manaraa.com

stack is decreased by 64 bytes. This annotation appendix is useful to shorten analysis-time and memory usage. Sub-function five is used for error handling only and thus it is not needed in general. So, the following annotation is made that avoids a contribution of sub-function five to the global WCET for a second WCET analysis:

```
snippet "sub_function5" is never executed;
```

Upper loop bounds for the functions that read out the characteristic map and line are annotated as well. The entry point of the analysis is the example function. No more annotations are necessary to get a first WCET by aiT.
When using aiT build 63339 and 70825, the determined WCETs of the example function using the manually build binary including debug information (size: 28 KiB) differs from that using the manually build binary without debug information (size: 10 KiB). Removing the compiler parameter *-g* results in higher WCETs. Version build 73031 does not suffer from that problem and provides the same results for the manually build binaries containing symbolic debug information or not. AbsInt Angewandte Informatik GmbH provided the exchange of the binary in order to investigate the problems occurred with versions build 63339 and 70825. This was not possible because the agreement of the legal department could not be obtained in that short evaluation time. So, the support team was contacted and a lot of time-consuming tests were made in order to try out several annotations and settings until aiT build 73031 was available, see section 2.6.3. So, aiT build 73031 is meant in the following when not stated otherwise. It determines a WCET of 11275 cycles for the example function to be investigated when including sub-function five into analysis and 2362 cycles when excluding it by the annotation described above. This high difference is caused by aiT's assumption that sub-function five might also be called by sub-functions one and two. This is illustrated by the call graphs shown at the end of the analysis, see figure 2.9 and figure 2.10. The red arrows within the graphs indicate the WCET path.

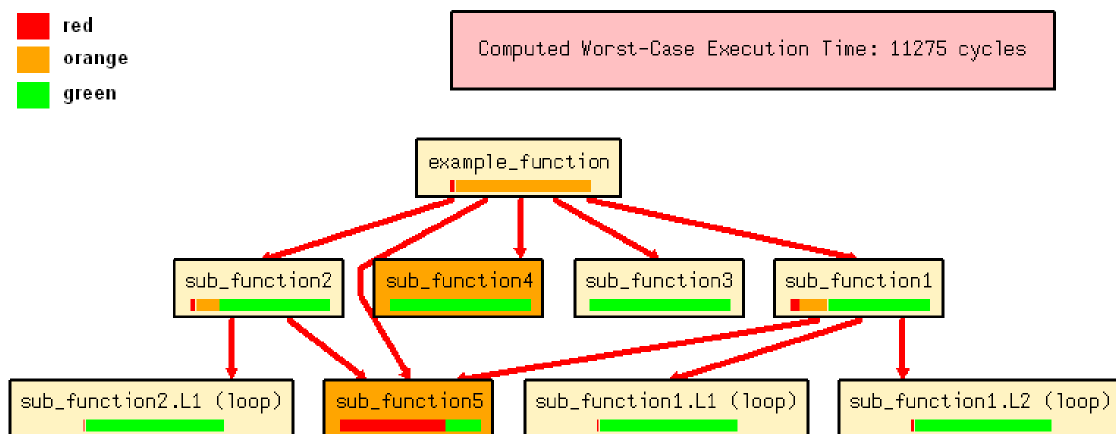



Figure 2.9: aiT's (build 73031) call graph analyzing the manually build binary including sub-function five



المنارة للاستشارات
www.manaraa.com

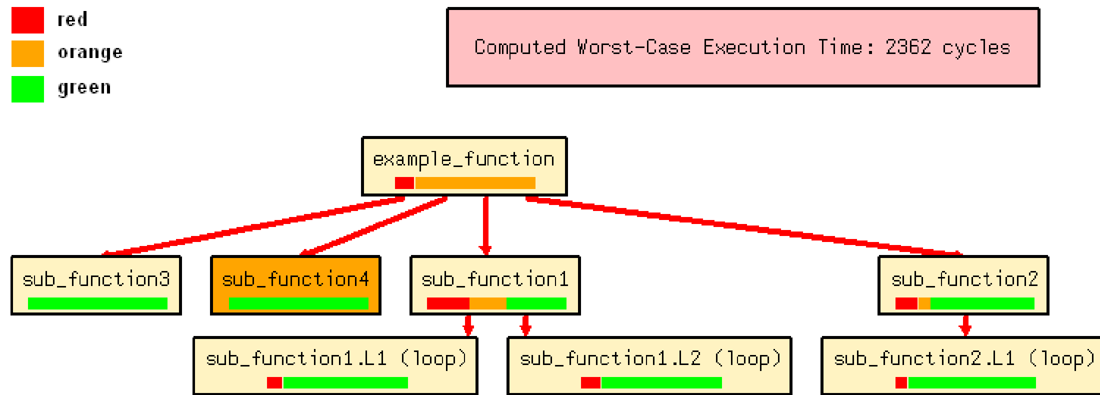



Figure 2.10: aiT's (build 73031) call graph analyzing the manually build binary excluding sub-function five

The red area of the bar within each call graph block indicates the corresponding function's contribution to the global WCET. The orange area stands for the portion of the corresponding function's sub-functions. A fully orange block indicates that its execution time is annotated. The call graph does not indicate how often a certain function is called. For example, sub-function five might be called twice by the example function. This and more analysis information can be gained by zooming in each block. The block's assembly code, the WCET path within the block and how many times a basic block is executed are listed. That "visualization" of the WCET might help to refine the analysis by additional annotations about infeasible code or to detect bottlenecks in the code.
The WCET including sub-function analysis is higher (2362 cycles) than that one with annotated sub-function WCETs (2340 cycles). The reason is that the overhead caused by the calls to the annotated sub-functions obviously is assumed to be too low. This is confirmed by removing the annotation appendix *"and does not violate calling conventions"*. Then, the WCET using annotated execution times is higher than that one including sub-function analysis. This phenomenon also does not occur when using local worst-cases for analysis. In the example, a WCET considering only local worst-cases is lower (2320 cycles) than the normally calculated one (2362 cycles). Table 2.1 summarizes the aiT results but does not list the results when removing the mentioned annotation appendix.

### Analyzing Final Binaries

Analyzing the final software-level binary is late in development process and might be used to determine the WCET of processes and tasks at once. It considers the final compiler settings and code optimizations. Two binary versions of a final software level are used: a big ELF-file containing the whole transmission software (size: 6.5 MiB) and a smaller one containing one part including the example function (size: 2.8 MiB). Both are compiled for TriCore 1766 and contain symbolic debug information.
When using aiT build 63339 or 70285, loading the big binary takes round eight minutes.



المنارة للاستشارات
www.manaraa.com

| function | source binary | using local worst-cases | including sub-function analysis | WCET [cycles] | analysis time |
|---|---|---|---|---|---|
| example fct. | mb | no | yes | 11275 (2362) | approx. 1 min |
| example fct. | mb | yes | yes | 11233 (2320) | approx. 1 min |
| example fct. | mb | no | no | 11255 (2340) | approx. 1 min |
| example fct. | mb | yes | no | 11255 (2340) | approx. 1 min |
| sub-fct. 1 | mb | no | yes | 3199 (1386) | approx. 1 min |
| sub-fct. 2 | mb | no | yes | 2269 (512) | approx. 1 min |
| sub-fct. 3 | mb | no | yes | 61 (61) | approx. 1 min |
| sub-fct. 4 | sb | no | - | annotated: 39 | - |
| sub-fct. 5 | sb | no | - | annotated: 1718 | - |
| example fct. | sb | no | yes | 8290 (2651) | 40 min |
| example fct. | sb | yes | yes | 7798 (2449) | approx. 1 min |
| example fct. | sb | no | no | 8136 (2590) | approx. 1 min |
| example fct. | sb | yes | no | 8004 (2511) | approx. 1 min |
| sub-fct. 1 | sb | no | yes | 1113 (1113) | approx. 1 min |
| sub-fct. 2 | sb | no | yes | 505 (505) | approx. 1 min |
| sub-fct. 3 | sb | no | yes | 141 (141) | approx. 1 min |
| sub-fct. 4 | bb | no | yes | 39 (39) | approx. 1 min |
| sub-fct. 5 | sb | no | yes | 1718 | approx. 1 min |
| example fct. | bb | no | yes | 8084 (2605) | 67 min |
| example fct. | bb | yes | yes | 7710 (2448) | approx. 3 min |
| example fct. | bb | no | no | 7969 (2570) | approx. 3 min |
| example fct. | bb | yes | no | 7833 (2495) | approx. 3 min |
| sub-fct. 1 | bb | no | yes | 1095 (1095) | approx. 3 min |
| sub-fct. 2 | bb | no | yes | 493 (493) | approx. 3 min |
| sub-fct. 3 | bb | no | yes | 142 (142) | approx. 2 min |
| sub-fct. 4 | bb | no | yes | 39 (39) | approx. 2 min |
| sub-fct. 5 | bb | no | yes | 1662 | approx. 3 min |

Table 2.1: WCETs determined by aiT for TriCore v2.0 build 73031, values in brackets are determined by using the annotation *snippet "sub-function5" is never executed;* (mb = manually build, sb = small binary, bb = big binary)



المنارة للاستشارات
www.manaraa.com

It holds for both final binaries that it is not possible to get a WCET of the example function when including sub-function analysis without setting the local worst-cases option. Otherwise, memory runs short during analysis on the test system. The analysis of the example function using the small final binary takes round 4.5 hours if annotating sub-function WCETs but not using local worst-cases for calculation. The annotated sub-function WCETs are determined by separate analyses. Even the analysis of sub-function one takes up to 1 hour when using aiT build 70825. The first both versions also fail on the analysis of sub-function five. The analysis re-iterates again and again during first analysis phase until memory runs short. The reason might be that this sub-function calls a lot of sub-functions that can be seen in the call graph of this function that is provided after analysis with aiT build 73031. As result, the memory usage explodes and the analysis terminates. The main reason for those problems is the bug that is fixed in aiT build 73031. Debug labels are taken as routine labels and a routine call affects the pipeline and cache behavior like described above. It is assumed that this leads to a state explosion and memory consumption increases to several gigabytes. In combination with the small amount of RAM of the test system, very long analysis times respectively analysis terminations due to memory shortness are the result.
Fortunately, the analysis of the example function using both final software-level binaries works well with aiT build 73031. Only in few cases the analysis takes quite long. It is supposed that it works better on a system with a plenty of RAM. The results are listed in table 2.1. It is remarkable that the analysis time and resource usage obviously depend on the size of the binary because the manually build binary does not suffers from that problem. Even when using aiT build 73031, the calculated WCETs of the several binaries differ. One reason might be different compiler settings that are used for the final and manually build binaries. These settings might change the code layout and therefor could lead to different WCETs. Analyzing sub-function four and five separately provides a WCET of 39 cycles and 1718 cycles (small binary) respectively 1662 cycles (big binary). These WCETs are annotated for sub-functions four and five when analyzing the manually build binary, like described in the antecedent sub-section. Again, the WCET including the sub-function analyses is higher (2651 cycles) than the one using annotated WCETs (2590 cycles). When removing the annotation appendix *"and does not violate calling conventions"*, aiT determines a WCET of 2805 cycles (small binary) when annotating sub-function WCETs whereas the WCET including sub-function analyses keeps 2651 cycles. Using local worst-cases provides the expected behavior without annotation adaption, see table 2.1. A more important observation is that the WCETs of the sub-functions do not change when excluding sub-function five from analysis. In contrast to the analysis of manually build binaries, aiT does not assume that sub-function five is called by sub-function one and two when analyzing the final binaries. This is indicated by the blue arrows within the resulting call graph, see figure 2.11. It is supposed that the value analysis works better on the final binaries because more information about register and memory contents is available. In the concrete example, the call to sub-function five by sub-functions one and two would be caused by incorrect information about the size of the



المنارة للاستشارات
www.manaraa.com

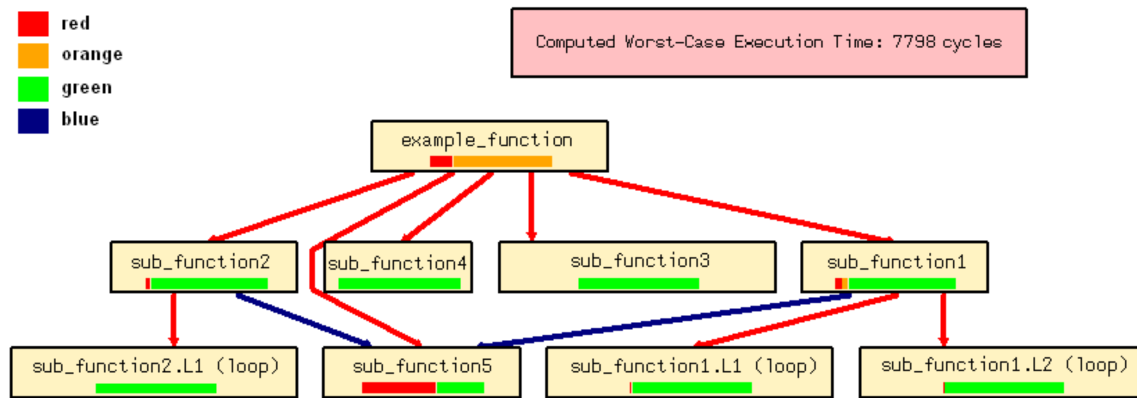



Figure 2.11: aiT's (build 73031) call graph after analyzing the small final binary using local worst-cases (the routines are renamed)

characteristic map respectively line. However, this information is assumed to be included in the final software-level binaries and is detected by the value analysis. So, the path analysis, which is performed by lp_solve, determines that a call to sub-function five and the WCET path are mutually exclusive with respect to the results of the value analysis. So, the WCET path does not contain sub-function five to be called by sub-functions one and two. Nevertheless, a review of the call graph and further annotations are necessary to check the correctness of the results. However, this and the user annotations require knowledge about the software's internals.

### Summary

The experiences made with aiT in scope of this work are comparable to the results of several case studies, see [8], [22] and [23]. These case studies describe the use of aiT and other tools in order to determine the WCET of embedded automotive software. The bigger and more complex the code snippets are, the more aiT annotations are necessary the more resources aiT needs. So, a lot of work has to be invested into the first analysis. Some annotations can be used again later. [22] describes five industrial case studies on timing analysis. In its fifth study, the results of static analysis with aiT for the Infineon C167CS processor, dynamic analysis with an in-circuit emulator and the common measurement technique are compared. The common technique uses operating system (Rubus) timing measurement functions and adds a safety margin in order to estimate the WCET. The over-estimation of aiT is between 4% and 33% in comparison to the emulator results. In all cases except one, the aiT results are much lower than the commonly measured values plus the safety margin, at least 38% and 59% in average [22, p. 7]. The WCET Tool Challenge 2006 [54] also evaluates several WCET tools and gives aiT good results.
For further work, it should be investigated, how aiT is capable to handle C++ code developed by ZF. It should handle this because of the programming restrictions described in section 2.6.1. A tool like aiT allows analyzing single functions, isolated from the rest of



المنارة للاستشارات
www.manaraa.com

the system. The tests show that an analysis of the binary of a final software level provides the most precise results because all compiler optimizations are included there. No extra hardware is necessary but the target microcontroller has to be supported. According to an AbsInt employee, a best-case execution time (BCET) analysis is kept in mind for further versions of aiT.
Due to the time limit of this work, the examination of aiT stops here. There will not be a deeper examination of the aiT internals and the reasons for the differences of the results. The referenced papers show that this could easily fill a whole diploma thesis. One has to keep in mind that aiT requires a lot of resources in the tested version and only certain microcontrollers are supported. However, it is assumed that aiT provides safe WCETs.

### 2.6.4 Tessy

#### Preface

Like described in section 2.5.5, Tessy is a unit-test tool for C-code with the possibility to perform unit-tests on the target. Tessy is used in version 2.5.9. The intention is to use this tool for WCET measurements of units, e.g. single functions. It is intended to measure all important paths through the code. Important means, all possible paths should be measured that are taken through normal operation. In particular, paths containing error handling possibly can be skipped. That is comparable to the exclusion of sub-function five by aiT. Test cases have to be specified to provide an input so that a certain path is taken on execution. The output is uninteresting unless it is intended to check the functionality of the function as well. This procedure gives the tester full control over the tests and possibly avoids rerunning time-consuming static WCET analyses due to analysis refinements.

#### Specifying Test Cases

The same code as for the examinations of aiT and the common techniques is used, see section 2.6.1. In order to reduce the complexity of specifying the test cases, each sub-function is analyzed separately. For sub-functions four and five, the source code is not available. The call to sub-function four is removed from the source code because it is assumed to take only 39 cycles, according to aiT (see table 2.1). The call to sub-function five is removed as well because it is for error handling only. Thus, it does not contribute to the global WCET. The resulting WCET are assumed to be comparable to that determined by aiT when excluding sub-function five from analysis.
Working with Tessy is as follows. The C-files are loaded and Tessy determines the contained functions. Then the interface of the function of interest (the test object) is edited. There, the required input and output variables to take the paths are determined. External variables and stub-functions also are defined by Tessy. Then, the test cases are defined. Each test case provides the input to take a certain path through the code. In particular, the intended example function has eight test cases. Error handling is assumed to do not occur



www.manaraa.com

during normal operation, so the short (error handling) path within the example function is not considered. The remaining tests cover the cases that could occur due to the three IF-statements within the example function. Sub-functions one, two and three are removed from it.
For sub-function one, twelve tests are specified. The first nine ones are needed to cover all cases that could occur when reading out the characteristic map. Each of its two dimensions causes three cases. A given parameter has to be searched within the map. It therefor can be below or above the lowest respectively highest value of the map or forces the read-out function to iterate until the right value is found. The last three of the twelve tests are the consequence of handling some special cases.
Sub-function two has three test cases. The first two cases occur when calling that function with a parameter that is below respectively above the lowest (highest) value of the characteristic line. Again, the third test forces this read-out function to iterate until the right one is found.
For sub-function three, four tests are specified because it contains only two IF-statements. The CTE helps to specify the test cases, especially when the number of paths respectively test cases grows. For each test case, the corresponding values can be specified within the CTE. So, the CTE makes systematic testing easier. Unfortunately it is not possible to import the created test cases with CTE v2.4 into Tessy v2.5 because of version conflicts.

### Execution Time Measurement

The automatic execution time measurement is only available for targets connected to a Lauterbach TRACE32 Debugger and for C166 microcontrollers.So, the combination of TRACE32 as target and Tasking TriCore VX as compiler is selected within the TEE (Tessy Environment Editor). The same compiler and additional files are given to Tessy as used in section 2.6.3. Tessy generates source code and builds an executable. Then, it tries to execute it on the target. Therefor, the executable is loaded to the target through a debugger script that is created by Tessy. Additional commands within this script, i.e. for specific target requirements, can be given through the module properties or the TEE.
First tests are made with the TRACE32 simulator in order to check that the communication between Tessy and TRACE32 works. Although the communication works fine, the simulator is not suitable for execution time measurement because it is only an instruction-accurate simulator. The runtime functionality of TRACE32 bases on the elapsed processor cycles. The simulator simply counts one cycle per executed assembler instruction and does not take caches etc. into account. The so determined "WCETs" are 17 instructions for the example function (without sub-functions), 24 for sub-function one, 14 for sub-function two and ten for sub-function three. Those are 65 in sum. These 65 assembler instructions do not reflect the real WCET because the overhead due to sub-function calls are not considered as well as a load instruction might need more or less processor cycles depending on if the data is in cache or not.
When trying to perform the tests on the ECU, it turned out that the TriCore 1766 supports



المنارة للاستشارات
www.manaraa.com

only two hardware breakpoints while Tessy needs three ones for automatic timing measurement. Thus, the timing measurement has to be included manually. It is intended to read out the timer register *STM0* of the processor before and after the call to the test object. This approach is very similar to the use of the timestamp counter of x86-based processors. However, the TriCore timer needs a complex initialization procedure. So, it is intended to get the execution of standard unit-tests on the target working properly before generating timing measurement code. Therefor, it is important to adhere strictly to the Tessy manual in order to get Tessy working with TRACE32 because the Tasking TriCore VX compiler has a lot of complex settings. In particular, the handling of the locator files requires a lot of expert knowledge.
A prototypical procedure to run the unit-test on the target is as follows. The host system running Tessy is connected to the TRACE32 debugger that is connected to the target ECU which is set up by a supplied debugger script. As Tessy builds a standalone application, the operating system and the application provided by ZF have to be deleted before. Otherwise, the operating system's control functionalities to supervise the application software cause resets when trying to run the unit-test. After loading the executable into the target using the standard script generated by Tessy, the execution goes only until the main function of the executable. Then, the communication between Tessy and the target fails because Tessy is not allowed to set hardware breakpoints. Even using compiler and linker options from the original software build process and declaring the whole memory of the target as flash in order to be able to set hardware breakpoints in RAM does not work. Due to the small amount of RAM (56 KB) and in order to provide realistic conditions, it is intended to flash the executable into the target's flash memory. The necessary flash commands can be given to the debugger script that is generated by Tessy in order to load the executable into the target. The executable has to be flashed to a certain memory region because the execution starts here after initialising the hardware with supplied code. This requires a memory mapping that locates the executables at this position. Unfortunately, the locator file that describes the memory mapping is supplied to ZF and could not be adapted adequately.
Further investigations of the locator file, the startup-code and eventually of the initialisation code have to be made in order to get Tessy working properly with the target ECU. Due to the limited time of this work, eventually very time-consuming attempts to get the unit-test running are left for future work. Even consultation of the Tessy support team did not lead to a working solution. It might be better to use an evaluation board instead of the target ECU because the hardware initialisation and memory mapping assumably are more transparent and could adapted more easily. Then it has to be considered that the conditions like processor frequency and memory access timings are equal to those of the target ECU.

### Summary

The advantages of this approach are that the tests are assumed to run quickly, no complex hardware model is needed and the tester will have full control over the test cases. The-



المنارة للاستشارات
www.manaraa.com

oretically, it is possible to determine a kind of BCET (which is not guaranteed to be the BCET). If it were possible to measure all paths through the code, a probability distribution of the execution times could be given.
Disadvantages are that it can not be guaranteed that the WCET is measured until not all possible inputs are measured. However, the coverage analysis provided by Tessy might help to cover all important paths. If there are a lot of paths through the code, the total time of the test will also grow. The approach using Tessy requires a complex preparation at all and it is restricted to C-code. Real hardware and the TRACE32 debugger are needed. As the functions are tested isolated from the rest of the system, hardware- and software-based interdependencies are not taken into account. Further, it is very hard to determine and to induce the worst initial hardware-state. In order to avoid too optimistic results due to cache and pipeline effects when executing the test-cases sequentially, code should be included which sets up a pessimistic initial hardware state before each run. The work with Tessy shows that typecasts can make source code changes for WCET determination necessary. In particular, if a pointer to a data structure is casted to a void pointer and given as input parameter, Tessy can not initialize the data structure.

### 2.6.5 Alternative Proposals

All previous described methods base on compiled code. If no code is available or execution times are required before the code is ready to compile, other methods are necessary. One possibility is the **ITA (instruction timing addition) method**. The CFG is build manually from the source code and the WCET of each basic block is calculated from the cycles of its instructions, like described by [19, p. 39 cpp.]. A basic block begins at the first instruction, a branch target or the very first instruction after a branch instruction. It ends at the beginning of the next basic block. Figure 2.12 illustrates this. The WCET of each basic block can be determined by the accumulation of the cycles each instruction in the basic block needs. This is a very hardware dependent approach but does not take delays caused by memory accesses or effects of caches and pipelines into account. Interdependencies between different tasks or the operating system are not considered. Methods to calculate the WCET of the program from the WCET of the basic blocks are described in [53, p. 14 cpp.] and in section 2.3.1. Here, a simple calculation method like the path-based one might be used.
In order to gain results if no code is available, **execution times could be derived from similar projects**. For example, the new hardware is faster than the old one and the new software's functionality is very similar to the old one. Very first execution time estimations can be calculated by just multiplying the old execution times with the ratio of old and new hardware speed. However, cross examinations of execution times, for example from SoftCar experiments running on the development system, are assumed to be insufficient respectively very imprecise because different hardware features of the processors like cache size, memory access time, pipeline depth, superscalarity or special floating-point performance could lead to different "conversion factors" depending on the code. The



المنارة للاستشارات
www.manaraa.com

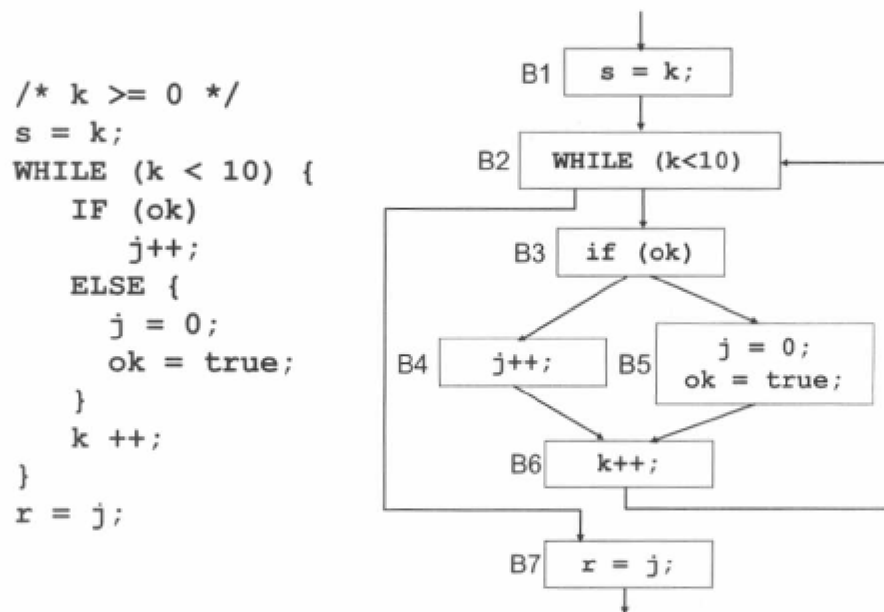



Figure 2.12: Example source code and related CFG for ITA, from [61, p. 2]

same holds for examinations based on analyses of other ECUs with another processor. So, deriving execution times from similar projects only works under the condition of using the same or very similar hardware. Another approach is to assume execution times of already implemented software with a similar functionality on the same hardware. However, often the software functionality and complexity change. Then, such an approach becomes not suitable respectively is very imprecise in general.

In some cases, the above described methods using static WCET analysis or unit-testing are not usable. If execution times are needed earlier than those determined by the common technique, a **manual unit-test approach** could be performed. A software frame calling the function to be tested and containing the measurement code is needed. Together with a suitable locator file and startup-code, a standalone application can be build which runs on the target, an evaluation board or on a cycle-accurate simulator. The required input has to be hard coded into the source code or given by the calling function. This is a time-consuming approach. Like all static WCET analysis and unit-test approaches, the interdependencies between other software components, the hardware and the operating system are not taken into account. The results are not guaranteed to be safe.

The use of a **hybrid WCET analysis** tool might be similar to the approach using Tessy. A divide-and-conquer strategy reduces the measuring complexity. Advantages of such a tool are the sophisticated calculation of the global WCET from these of the basic blocks and a higher automation level. An investigation of this tool using a trial version could be topic of further work. Hybrid approaches like MTime (see section 2.5.4) suffers from the same problems of dynamic WCET analysis. The measured execution time is not guaranteed to



المنارة للاستشارات
www.manaraa.com

be the WCET.
**Approaches to make WCET analysis easier** is to use simpler hardware, e.g. without (multiple) caches. Another possibility is to restrict the programming style. Loops should have constant iteration numbers. Recursion should be avoided when possible or have a constant depth. In general, the execution time of the code should depend as few as possible on input parameters and should be as constant as possible. Such code might perform each computation and there are only very few paths through the code. At the end, only the necessary results are assigned to the appropriate (global) variables. This approach also makes it less difficult to determine the BCET of certain code. It reduces effects of large jitter which could effect end-to-end paths within the system, like described in section 3.5. Programming features like dynamic binding or dynamic memory (de)allocation should not be used but are not used by ZF anyway. They would cause delays that can not be determined statically. The programming language has to be chosen in order to keep the ability to use a certain WCET analysis method. Code with only a few possible paths is easier to analyze respectively it is easier to specify test cases for it. If the code should not be analyzed by the software developer, the code has to be well commented too in order to reduce questions to him later.

### 2.6.6 Result Analysis

Table 2.2 summarizes the (worst-case) execution times gained using the described methods. One has to keep in mind that the tested aiT version is for the TriCore 1796 whereas the target processor is a TriCore 1766. The WCET of 0.03314 ms results from the clock frequency of 80 MHz and the 2651 cycles determined by aiT without considering the error handling sub-function five (see table 2.1).

| method/tool | WCET [ms] | WCET analysis approach |
|---|---|---|
| analyzing hardware-traces | 0.0022 | dynamic |
| INCA measurements | 0.08105 | dynamic |
| aiT | 0.03314 | static |
| Tessy | 65 assembler instructions | dynamic |

Table 2.2: Summary of the obtained WCETs using different methods respectively tools

Both processors (1766 vs. 1796) have differences like the clock frequency (80 MHz vs. 150 MHz), internal flash (1.5 MB vs. 2.0 MB) or SRAM size (72 bytes vs. 192 bytes). However, both have a similar hardware architecture and a superscalar pipeline which allows processing of round 1.3 instructions per cycle. So, the result provided by aiT is assumed to be suitable to determine the WCET of the given code for the TriCore 1766.
Surprisingly, the WCET gained by the INCA measurement is the highest one. Even the



المنارة للاستشارات
www.manaraa.com

one determined by aiT is lower. However, the lowest execution time measured by INCA is about 0.014 ms and so it is below the aiT result. It is assumed that the high execution time is caused by interrupts which are not filtered plus the measurement overhead. In contrast, the execution time gained from the hardware-traces is much lower than the other ones. The reasons might be that the hardware-trace is very short so that only a average execution time was measured and it was recorded while normal operation and not while running a 3-minute SoftCar RT-script. Further, the execution time of 0.0022 ms is the pure execution time. Interrupts and the INCA measurement overhead are not taken into account. Section 3.4 confirms these results but shows that the differences do not scale that much like here. So, it is assumed that especially for small code pieces the measurement by INCA might provide too pessimistic results. Unfortunately, for the Tessy approach no results are available. Just the number of assembler instructions could be gained by using the TRACE32 simulator. It is about 65, see section 2.6.4. In general, the determined WCETs highly depend on the measurement technique used. So, an evaluation of the results always has to be made carefully.

## 2.7 Measuring And Processing Execution Times

The question if a separation of coding and WCET analysis makes sense is not easily answered. The **common technique** implicitly separates both steps to different employees. The software developers check the functionality of their written code within a software-based environment. The software integration expert builds the final software-level and performs several functionality tests. Writing SoftCar RT-scripts that are suitable for WCET analysis requires knowledge about software internals. Because the software is a black-box, it is practically impossible to write a SoftCar RT-script respectively to stimulate the software another way in order to induce the worst-case situation where a certain task or process has its WCET. So, the common technique is a kind of black-box test that separates coding and measurement.
When using **aiT**, the software developer has the ability to analyze his code on his own development system. However, annotations concerning the hardware that need hardware expert knowledge have do be made. For precise results, the hardware expert has to work together with the software developer until the tool is configured completely. It is not possible to perform real black-box tests both for software developer and hardware expert.
In case of **Tessy**, separation of test case specification and measurement is assumed to be more practicable. The software developer might give the test cases and code to a tester which performs the preparation and measurements of the test on the target. This eventually burdens the software developer with extra work. But he might be the only one to specify the test cases quickly and correct. However, specifying test cases is usual for several projects within ZF. So, test the specification has to be extended to WCET analysis. The tester needs a lot of knowledge in order to provide a test frame that consists of the right configuration of the hardware and software. This has to be done once and is comparable



المنارة للاستشارات
www.manaraa.com

to hardware annotations when using aiT.
When **processing the WCETs**, one has to take into account that Tessy and aiT only provide the WCET of certain code snippets like a single function. Therefor, the global WCET of whole processes and tasks has to be calculated from these values. These local WCETs are assumed to be safe, in particular statically determined ones. A non-consideration of software-sided interdependencies like mutual exclusion of WCETs of different processes potentially leads to a too pessimistic global WCET. If the software is a black-box, it is even not possible to determine the case-sensitive worst-case combination of execution times for a certain task or process. Further, the non-consideration of hardware-sided interdependencies potentially leads to a too optimistic global WCET. Such effects might occur due to a dirtied cache, pipeline stall or an incorrect branch prediction in consequence of interrupted code execution or a function call within the code. This could lengthen the WCET of a code snippet because WCET analysis just determines the WCET of the code under the assumption that it is executed as a whole. It is unknown if the simple accumulation of WCETs provides a safer result than an exhaustive WCET analysis that takes all (hardware-) effects into account, which assumably is not possible due to the black-box character of the system.
Tasks with a short cycle time that run on an ECU might include supplied software (at all). This is dependent on the concrete project. These software parts are a black-box for ZF. They make a WCET analysis in early development phase difficult. So, only a measurement-based or maybe a static WCET analysis of these processes is possible using the final binary. This is late in development process. It should be investigated if it is possible to get WCETs from the supplier or to build an "empty" software level containing only this supplied software. Then, analyzing them could be done simultaneously to the coding phase of the ZF-software. Another possibility is to assume experience-based execution times for these processes.

## 2.8 Summary

This chapter gives an overview of possibilities to determine (worst-case) execution times. The common measurement-based execution time determination used by ZF easily provides execution times of processes and tasks. It takes interdependencies between the measured code and other software components, hardware and the operating systems implicitly into account. A further measurement based WCET analysis approach uses a unit-test tool. It might reduce the complexity and quickly provides a WCET estimation of small code snippets like single functions. The use of the commercial static WCET analysis tool aiT and more ways to gain WCET estimations and proposals to make the analysis easier are described too.
The extra effort that is needed for a WCET analysis early in development process is significant. It also turns out that black-box testing is not suitable if very accurate results are needed. The described approaches using aiT or Tessy provide first WCET estimations of



المنارة للاستشارات
www.manaraa.com

single code snippets and maybe whole processes or tasks and move that from the system integration phase to the module-test phase of the V-model. The ITA method provides first WCET estimations in implementation phase but the results might be very imprecise. The WCET problem is a trade-off between effort and safety, like described in [22, p. 1]:

> "It is really a business issue, where the cost of a timing-related failure has to be weighed against the cost of various means of preventing or handling such a failure."

The described approaches focus on software provided by ZF. Like described in section 2.7, further software parts which are not developed by ZF have to be taken into account as well. It has to be investigated how to get precise WCETs of the supplied software early in development process. Further, section 2.6.3 describes that the statically determined WCET of a small code snippets depends on the development phase in which they are analyzed. Using compiler- and linker-settings of the final build process probably provides more precise results for early estimations. This section also partly invalidates the assumption that an accumulation of WCETs leads to a too pessimistic global WCET. Thus, an analysis of the final system is necessary for very precise results. When making a decision about the approach to take respectively which effort to invest, the following aspects should be considered besides the above mentioned statement. How critical is the violation of a deadline? Is it sufficient to work with WCET estimations? Does a static WCET analysis tool support future hardware platforms?



المنارة للاستشارات
www.manaraa.com

# 3 Scheduling Analysis with SymTA/S

## 3.1 Preface

The scheduling analysis tool SymTA/S is developed by Symta Vision [43], a split-off of Technical University of Braunschweig. SymTA/S stands for **Sym**bolic **T**iming **A**nalysis for **S**ystems. In the following, SymTA/S v1.3 is described. It calculates worst-case response times (WCRTs) for tasks on a single processor, for messages on a shared bus and for end-to-end paths in a system, e.g. from a sensor via an ECU to an actor communicating by a bus. A WCRT of a task is the time between task activation and task termination. So, the worst-case execution time (WCET, described in section 2.1) of the task is not the only crucial factor for it. The task's WCRT might be lengthened by preemptions, blocking delays on an exclusive resource or interruptions by ISRs, for example. Besides these scheduling and end-to-end timing analyses, SymTA/S provides a sensitivity analysis and an optimization function for processors as well as buses. So, it is intended to be used for:

- ECU scheduling verification, timing budgeting and optimization
- bus scheduling verification and optimization
- system scheduling verification and optimization

In contrast to timing tests or simulation of a system, scheduling analysis *calculates* best- and worst-case situations respectively WCRTs. Therefor, it is assumed that safe results are provided. However, the results might be too pessimistic because the calculated situation does not occur in reality. Because of that, SymTA/S visualizes the worst-case situation in order to check the results. A further advantage of scheduling analysis is the possibility to easily examine system parameter changes. Time-consuming tests of the system behavior could be determined before new code is implemented. Such verifications are necessary, primarily in complex systems where the effect of a parameter change is not detectable at first sight. For example, the reduction of a task's WCET might lead to a higher bus load that could affect the whole system behavior.
SymTA/S provides analyses for scheduling strategies of operating systems and bus systems typically used in automotive industry. This includes several OSEK variants and AUTOSAR OS as well as analysis of bus systems like CAN and FlexRay. The following section introduces the OSEK OS standard and a specific implementation.



المنارة للاستشارات
www.manaraa.com

## 3.2 Automotive Software Example using OSEK OS

### 3.2.1 OSEK OS Standard

OSEK (Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug) is a registered trademark of Siemens AG. One of the most important specifications contrived by the OSEK respectively OSEK/VDX standardization committee [35] is the embedded operating system OSEK OS, often treated synonymously with OSEK. Further OSEK specifications are a uniform communication environment for automotive control unit application software (OSEK COM), a network management system (OSEK NM) or the OSEK implementation language (OIL). Some OSEK parts, including OSEK OS v2.2.1, are standardized by ISO (number 17356). There exist several commercial OSEK conform operating system implementations like osCAN, RTA-OSEK and ERCOSEK (by ETAS) as well as ProOSEK. HSE Free OSEK [27] is an open source implementation of an OSEK conform operating system by the University of Applied Sciences Esslingen.
According to its specification [34], OSEK OS is a **multitasking capable single processor operating system** for embedded systems. In order to keep application software portable to different ECUs, it has standardized interfaces between application software and operating system, e.g. service calls, type definitions and constants. The operating system is compiled and linked together with the application software. All resources and tasks are specified statically.
The operating system specification introduces interrupts and tasks. A tasks provides the framework for the execution of functions [34, p. 16]. There are three processing levels, illustrated by figure 3.1. The **priorities of tasks and interrupts are specified statically** and increase with the number. Interrupts have higher priorities than tasks and can always preempt them. Tasks are scheduled according to their priority which is described later. The OSEK OS specification distinguishes two task types: basic tasks and extended tasks. A basic task only leaves the processor when it terminates, the scheduler switches to a higher priority task or an interrupt occurs. An extended task additionally can leave the processor by entering a waiting state. Figure 3.2 illustrates the task state model of an OSEK OS extended task. State transitions *terminate* respectively *activate* are caused by a system service, e.g. when a task requests an unavailable resource respectively it gains the requested resource. Transitions *start* and *preempt* are caused by the scheduler and represent the assignment respectively release of the processor. An extended task can enter the waiting state by a system service, e.g. in order to wait for an event like an interrupt. The counterpart of this *wait* transition is the *release* transition. It causes a waiting extended task to enter the ready state and is taken when at least one event the task waited for occurred. The task state model of basic tasks differs in the absence of the waiting state. So, extended tasks provide more synchronization points than basic tasks. On the other hand, they require more resources (RAM).
According to the features of the OSEK OS implementation, it can be classified in one of four so-called **conformance classes (CCs)**. These CCs differ in the supported task types,



المنارة للاستشارات
www.manaraa.com

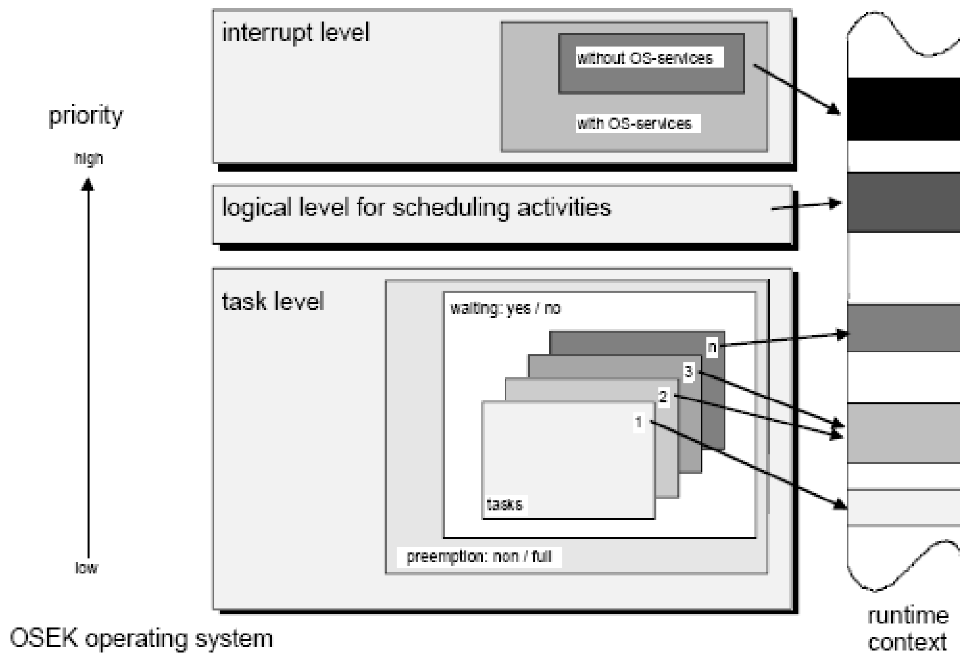



Figure 3.1: OSEK OS processing levels, from [34, p. 12]

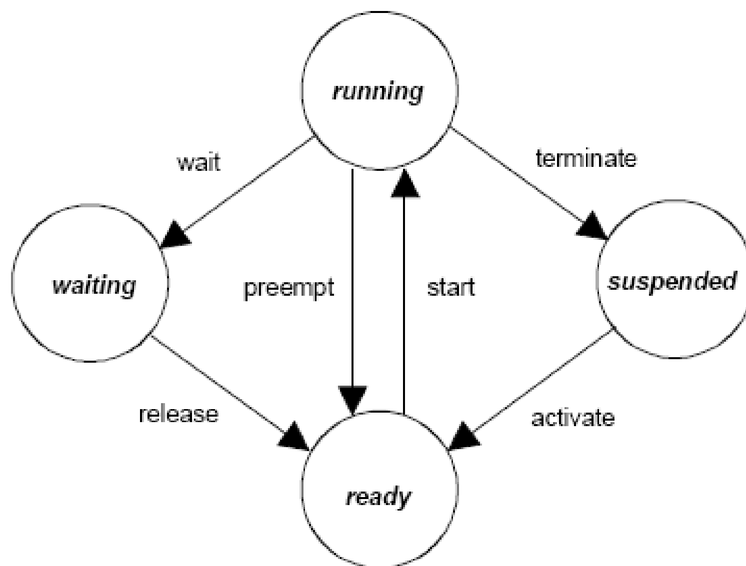



Figure 3.2: OSEK OS extended task state model, from [34, p. 17]



المنارة للاستشارات
www.manaraa.com

the possibility of multiple activations of a task and the number of tasks per priority. The simplest CC is BCC1 (Basic CC 1). It supports only basic tasks with different priorities and 1 activation per task. In contrast, BCC2 supports multiple tasks per priority and multiple activations per task. ECC1 and ECC2 (Extended CC) are corresponding with the difference that they support basic and extended tasks.
An OSEK OS task is activated, i.e. set to ready state, if no resource except the processor is missing by the system calls *ActivateTask()* or *ChainTask()*. The former one might be called by an Interrupt Service Routine (ISR), amongst others. Task termination is performed by the system calls *TerminateTask()* or *ChainTask()* that have to be called by the task itself. The latter one terminates the running task and activates a certain task. The **scheduler** determines the next ready task to be executed, i.e. set to running state, on the basis of the priority of the tasks. Tasks with the same priority are ordered in a kind of FIFO queue according to their activation order. A preempted task is set as the oldest task in the queue. A released waiting extended task is set as the newest task in the queue. The scheduler determines the oldest task which is in the ready respectively running state and which has the currently highest priority. So, low priority tasks might starve if always new high priority tasks are set to the ready state. Figure 3.3 illustrates the working principle of the OSEK OS scheduler. A call to *ActivateTask()* or *ChainTask()* enqueues a task in its corresponding queue.

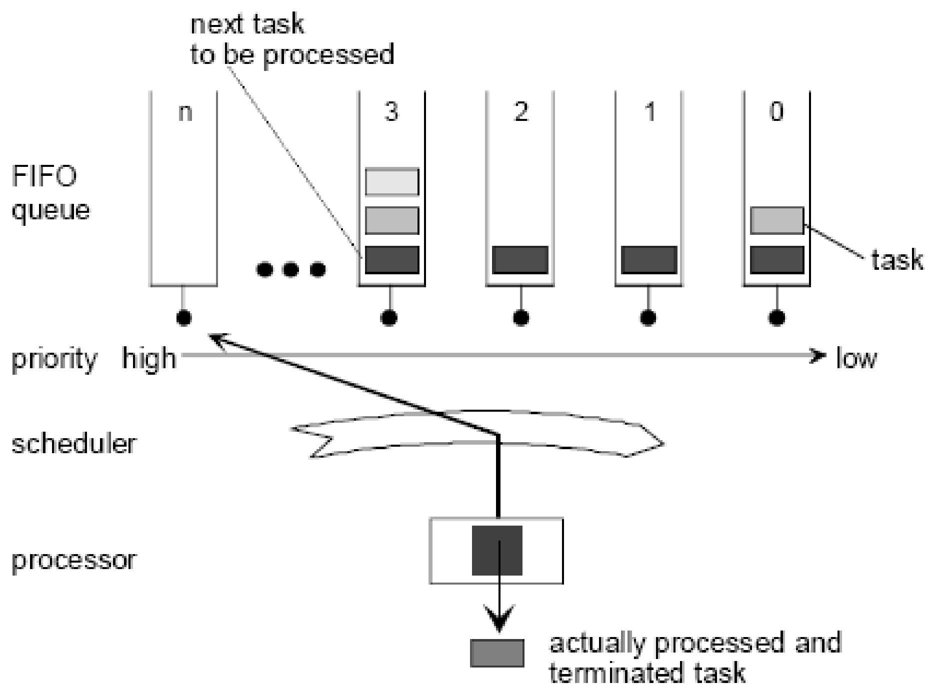



Figure 3.3: OSEK OS scheduler, from [34, p. 20]

OSEK OS supports different **scheduling policies**. The *full preemptive scheduling* allows



المنارة للاستشارات
www.manaraa.com

the preemption of a running task at each instruction. A preemption is caused when a higher priority tasks becomes ready or an ISR is executed. Re-scheduling occurs when:

- the scheduler is called explicitly
- a task terminates or is activated
- an extended task enters or leaves the waiting state
- a resource is released
- an ISR returns

During execution of an ISR, scheduling is disabled, see figure 3.1. The *non-preemptive scheduling* allows the re-scheduling only at certain points:

- task termination
- explicit call to the scheduler
- entering the waiting state (extended tasks)

OSEK OS provides the possibility to create non-preemptive groups of tasks. Tasks within this group can only preempt each other at certain re-scheduling points. Tasks with a higher priority than the highest priority task within the group can preempt every task of the group at any point. One way to create such a group is to use a so-called (exclusive) internal resource. This is no physical resource and has to be assigned to a task at system generation. An internal resource is taken by a task when it enters the running state and is released at a point of re-scheduling but not when it is preempted. The OSEK OS standard prescribes the **Priority Ceiling Protocol** (PCP). So, tasks that share such an internal resource are known as so-called *cooperative tasks*. Further, the PCP avoids deadlocks due to inconvenient resource requests. According to the PCP, a cooperative task could inherit the priority of the highest priority task within the group and therefore its priority is not static at all.
Imagine an OSEK system with periodic tasks and only one task per period. The highest priority is set according to the shortest period and likewise the lowest priority to the longest period. In case of full preemptive scheduling policy and no exclusive resources, the OSEK system behaves like a Rate-Monotonic Scheduling (RMS) system without especially set deadlines. In fact, the OSEK OS standard does not consider the specification of deadlines. The OSEK OS standard just provides the general conditions to implement an embedded real-time operating system.
The OSEK OS standard also allows different application modes of the operating system. Each mode can have different task sets and interrupts. It is also possible that different application modes share certain tasks or interrupts. A switch between these application modes at runtime is not supported in the OSEK OS standard.



المنارة للاستشارات
www.manaraa.com

### 3.2.2 OSEK Implementation in Automotive Industry

OSEK OS implementations used by ZF are RTA-OSEK, ERCOSEK or ProOSEK, depending on the concrete project. The following explanations base on a certain automatic transmission project. The corresponded ECU including a Renesas SH72 microcontroller contains amongst others the following software parts: the initialization code and the operating system from a supplier and the application software provided by ZF (this might differ in other projects). The operating system kernel is configured to provide a fast interrupt and several tasks with certain periods: amongst others 10ms, 20ms, 30ms, 50ms, 100ms and the background task. The used OSEK OS kernel is configured to BCC1. Thus, there are only basic tasks that can be activated only once per period. All tasks are set to preemptive scheduling. These task are activated through a call to *ActivateTask()* within the fast interrupt. So, each task switch causes an overhead which has to be taken into account by a scheduling analysis if possible. Further, possible activation offsets are restricted to a resolution of the fast interrupt.
An (activation) offset is a startup delay of a task within a group of synchronized tasks, e.g. tasks on the same ECU. If the offsets of all tasks are zero, they are activated at the same time. So, the introduction of offsets might decrease the number of preemptions or blocking time of a task. A former ZF project using a different ECU has a higher resolution because certain functionality had to be realized in software. This provides a higher task activation resolution but causes more operating system overhead. The offsets depend on the concrete project.
The periodic tasks of the automatic transmission project are divided into so-called processes. These processes in principle are calls to functions that call further functions. The execution order of the processes is hard coded. The execution and timing related control is performed by the so-called process flow control. The processes of the 100ms-task up to the 10ms-task share a common internal resource which is assigned to these tasks statically. So, these tasks perform cooperative scheduling and can not preempt each other. This is facilitated by the implemented PCP in the used OSEK OS implementation. Within this task group, re-scheduling is only possibly on process boundaries. When a process finishes, a flag indicates whether a further task of the cooperative group is activated. In this case, the scheduler is called and a higher priority (cooperative) task might be executed. The internal resource is handled automatically. However, each task and process of this group might be preempted by the interrupts or tasks with a priority higher than the 10ms-task. So, a scheduling analysis has to take these blocking times due to this cooperative scheduling into account.
The software running on the ECU distinguishes between different so-called operation modes (OM) according to the actual condition of the automatic transmission. These OMs are determined by the operation mode manager (OMM), a software subsystem. The operating system itself provides only one application mode because a switch between application modes at runtime is not possible. The current implementation has amongst others following OMs: initialization, cruise, limp home and shutdown. The actual one depends



المنارة للاستشارات
www.manaraa.com

on conditions like ignition state or position of the transmission selector lever and is determined by the OMM which causes OM transitions if necessary. Such a transition is only possibly at a certain task. It causes overhead because a special transition function has to be called which initializes all task for the new OM. The used OSEK OS implementation supports only 32 tasks. Therefor, different OMs share them. This is realized by providing so-called process tables. They contain several processes which are executed by a certain task in a certain mode. Further, the appropriate process tables are determined by a so-called container table. There exist one container table per task and OM. According to task and OM, the OMM determines the container table for each task which contains pointers to the process tables so that the right processes are executed.
The control of the timing constraints during operation is performed by the process flow control. That highly depends on the concrete project. A deadline in the broadest sense is determined by the automatic transmission mechanics. In a fictitious example, a gear switch would take several seconds and the hydraulic pressure to control the clutches would be calculated by the 10ms-task. If the 10ms-task would not finish before it is activated the next time, this would possibly lead to a less comfortable gear switch due to a higher clutch pressure step. The task would not be activated subsequently because the OSEK OS implementation is configured to BCC1. So, it would be tolerable that the 10ms-task misses one activation. If it would not be finished at the next but one activation, an error handler would be called. Depending on the concrete error, error handling could lead to an ECU reset or the software switches to the limp home mode. There are no hard deadlines within the system in the proper meaning of deadlines. The process flow control also controls the activation (order) of processes. For example, some processes are executed only every second or third time the correspondent task is executed. In general, it is not easy to answer if an automatic transmission is a hard real-time system and if harder timing constraints are necessary or if it has to be fault-tolerant in some cases. A rarely occurring less comfortable gear switch possibly is more acceptable for the driver than a blinking control lamp within the cockpit. In general, error handling and several mechanisms, e.g. how the cooperative scheduling is realized, depends on the project.

## 3.3 SymTA/S Internals

### 3.3.1 General Structure

The **general structure** of the SymTA/S tool suite is illustrated by figure 3.4. The core component is the analysis engine. It provides all basic scheduling analysis functionality. The analysis of certain resources is realized by component libraries. They are provided by Symta Vision or third party developers. Currently, libraries are available for several OSEK variants, e.g. RTA-OSEK and a generic OSEK OS, AUTOSAR OS, CAN and FlexRay as well as for further scheduling policies like RMS, earliest deadline first and time division multiple access. These libraries contain specific information about necessary scheduling



المنارة للاستشارات
www.manaraa.com

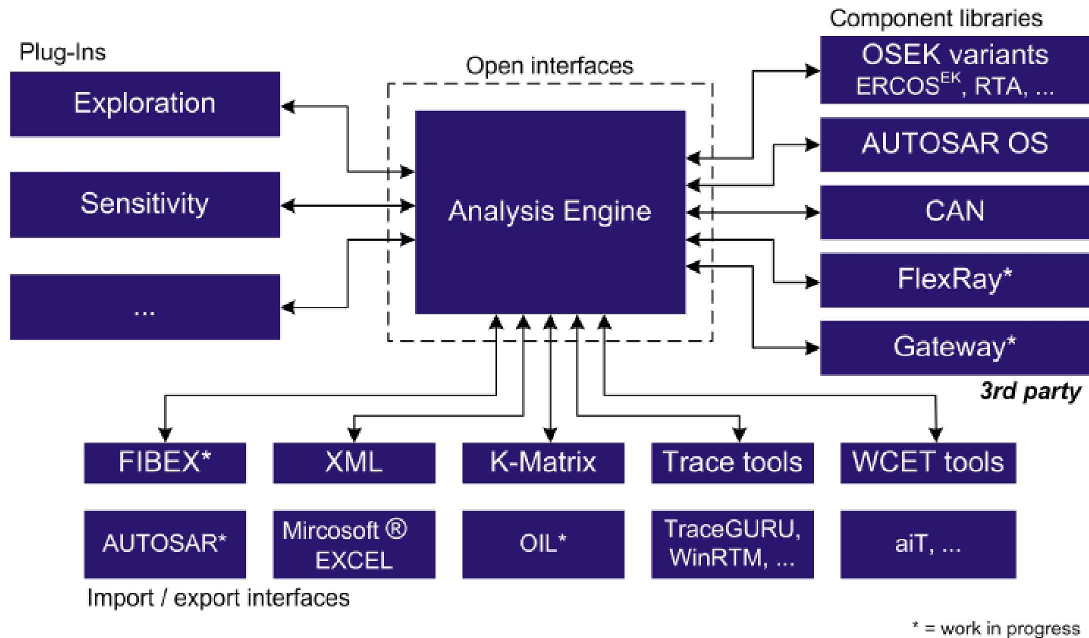



Figure 3.4: SymTA/S tool suite, from SymTA/S Main manual v1.3 page 16

parameters like task period, frame size or priority. SymTA/S allows creating own libraries. The **input parameters** for scheduling analysis are the scheduling policy, the BCETs and WCETs of the tasks/processes running on the processor, the size of messages on a bus and the activation frequency of the tasks respectively messages. In order to precise analysis, further parameters can be given. These are activation offsets (described in section 3.2.2), further assumptions about (dynamic) activation of bus messages, operating system scheduling overhead, general blocking delay or the use of an exclusive resource. Especially the WCETs are very important input parameters. Using lower WCETs than the real ones could lead to a positive scheduling analysis result but might cause problems during later system operating. Using too high estimated WCETs could lead to a negative analysis result that could not occur in reality. Because of that, chapter 2 is devoted to execution time analysis. SymTA/S also provides import interfaces in order to easily import systems respectively component descriptions like illustrated in figure 3.4. Unfortunately, an import interface for files stored in the description language format of OSEK (OIL) is not implemented in v1.3. However, it is possible to write and include own import scripts. An integration of WCET analysis tools like aiT (see section 2.5.1) is imaginable.
The **output** of SymTA/S are the best-case and worst-case response times of each task, bus messages and specified end-to-end paths as well as bus and ECU utilization. The WCRTs can be checked against specified deadlines. Further, the worst-case situation of each task, message and path can be visualized by Gantt-charts. The time axis within these charts is relative to the worst-case situation. So, it starts with zero at the beginning of the situation. SymTA/S allows generating a report file after analysis. This file contains all above de-



المنارة للاستشارات
www.manaraa.com

scribed information and parameters of the system and its components plus the mentioned Gantt-charts. It can be stored in common office formats.

### 3.3.2 Compositional Scheduling Analysis

SymTA/S follows the principle of **compositional scheduling analysis**. Figure 3.5 shows a very simple system that is analyzed by SymTA/S. It helps illustrating the implemented analysis approach. The OSEK-tasks (e.g. *OSEK-Task1*) and CAN-frames (e.g. *CAN-*

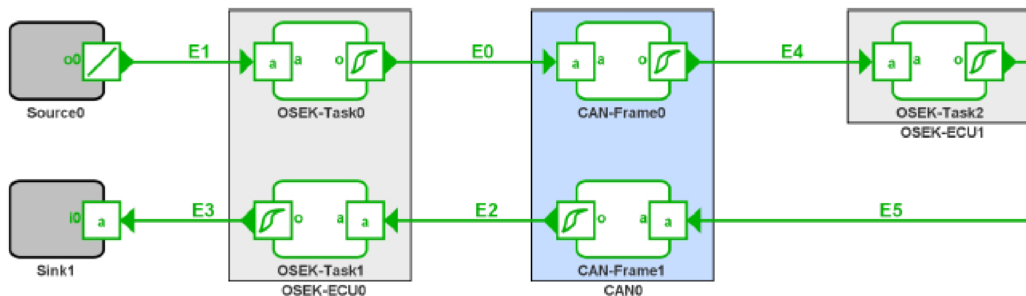



Figure 3.5: SymTA/S compositional scheduling analysis approach illustration

*Frame0*) are so-called *application elements*. They are mapped to corresponding resources, i.e to an ECU with OSEK-scheduler (e.g. *OSEK-ECU0*) or to a CAN-bus (e.g. *CAN0*). The arrows between them (e.g. *E4*) are so-called *event streams* that interconnect the application elements. SymTA/S distinguishes between activation models and execution models. Execution models are created by analyzing the execution elements locally on a resource using the libraries described above. In order to perform this local analysis precise, information about the activation of an execution element is needed. This information is transported by the event streams. These event streams can be used to carry activation information as well as data-related timing information. The output event stream of a certain application element propagates a certain *event model*. This event model is the input event model for the connected application element. So, analysis can be seen as flow-analysis problem that can be solved iteratively using event stream propagation, see [45, p. 3]. SymTA/S provides six event models. They are kept simple. So, not every possible activation scenario can be modeled. These models are self-contained, i.e. each possible change in the parameter values leads to one of the six models. Further, the parameters are simple enough in order to be used by the common local scheduling analyses. So, three parameters are used:

- period $T$
- jitter $J$



المنارة للاستشارات
www.manaraa.com

- minimum distance between two events $d$

In terms of SymTA/S, a jitter is a possible delay of a periodic event [44, p. 23]. If the jitter exceeds the period, theoretically two events can occur at the same time. This can not happen in reality and therefor, the parameter $d$ specifies a minimum distance between two events. There are two classes of events, the periodic and the sporadic ones. Each class has three models: simple, with jitter and with bursts. Figure 3.6 shows the lower and upper event functions for each model. The lower event function $n^-_{act}(\Delta t)$ determines the minimum number of events that could occur during a time interval of $\Delta t$. The upper event function $n^+_{act}(\Delta t)$ determines the maximum number of events that could occur during $\Delta t$.

| model | params | constraints | both<br>$n^+_{\text{act}}(\Delta t)$ | periodic<br>$n^-_{\text{act}}(\Delta t)$ | sporadic |
|---|---|---|---|---|---|
| simple | $<T>$ | $T > 0$ | $\left\lceil \frac{\Delta t}{T} \right\rceil$ | $\left\lfloor \frac{\Delta t}{T} \right\rfloor$ | 0 |
| w/ jitter | $<T,J>$ | $T > J \geq 0$ | $\left\lceil \frac{\Delta t + J}{T} \right\rceil$ | $\max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right)$ | 0 |
| w/ burst | $<T,J,d>$ | $J \geq T > 0, \quad d \geq 0$ | $\min\left(\left\lceil \frac{\Delta t + J}{T} \right\rceil, \left\lceil \frac{\Delta t}{d} \right\rceil\right)$ | $\max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right)$ | 0 |

Figure 3.6: SymTA/S event models, from [39, p. 4]

In some cases it is necessary to convert an event model into another one. The reason might be that a certain application element requires a certain input event model, e.g. an actor has to be activated periodically and not sporadically. In general, there are two possibilities to hold the requirements. First, the event model interface method formally converts an event model into another, e.g. a periodic one into a sporadic one. A further possibility is to force the conversion of an event model into another one. So, the conversion of a sporadic event model into a periodic one can be reached by introducing a little buffer, a register in terms of SymTA/S. Some of these transformations are lossless, some are lossy. See [39, p. 4 cpp.] for detailed information.
Figure 3.5 illustrates an example where *OSEK-Task0* is periodically activated by the external source *Source0* with a period of 10. This value might stand for milliseconds, but SymTA/S does not use units. So, the input event model for *OSEK-Task0* is simple periodic, carried by event stream *E1*. The tasks on *OSEK-ECU0* are scheduled preemptive whereas *OSEK-Task0* has the higher priority. Before analysis, the output event model of *OSEK-Task0* and so the input event models for all other application elements are unknown because they are not activated by an external source. In order to locally analyze *OSEK-ECU0*, an input event model for *OSEK-Task1* is needed. This is called a cyclic scheduling dependency [45, p. 6]. SymTA/S propagates the known event models along the path until each application element has an input event model. Then local scheduling analysis can be performed which might change the event models. So, local analysis has to be repeated, the event models propagated and so on. Analysis stops when the event models converge or a timing constraint is violated. This approach works because scheduling does not change an



المنارة للاستشارات
www.manaraa.com

event model period. Furthermore, the jitter along a path without buffering elements just increases, because each application element adds a delay to an event that goes through the system along a certain path of event streams.
In the example, *OSEK-Task0* has a response time between 3 and 4.5. So, it adds a jitter of 1.5 to its output event. Therefor, it has an output event model of $T = 10$ and $J = 1.5$. This is the input event model for *CAN-Frame0*. Local analysis determines a response time between 0.432 and 1.068 and now the output event model is $T = 10$ and $J = 1.5 + (1.068 - 0.432) = 2.136$. The described approach has to take more issues into account. In order to avoid too pessimistic results, the synchronous timing behavior along end-to-end paths has to be considered. The given example is straight forward because all elements are on a full event-triggered path. So, they are implicitly synchronized to each other, i.e. they are within a synchronized group. End-to-end analysis is described in detail by section 3.3.5.

### 3.3.3 OSEK OS Scheduling

In order to create an OSEK OS-ECU within SymTA/S, a so-called *COM-CPU* has to be created. One or more tasks can be added and mapped to the COM-CPU. For each task, the priority, best- as well as worst-case execution time, the scheduling policy (e.g. preemptive or cooperative) and possible scheduling overhead have to be specified. Further, processes and their BCETs as well as WCETs can be added. If choosing the cooperative scheduling policy and declaring a shared resource, the involved OSEK-tasks preempt each other only on process boundaries, like described in section 3.2.2. Figure 3.7 illustrates an OSEK-task mapped to a COM-CPU. The task is activated by an external source with a period of 10 which is connected to the input port *a*. A task supports only one activation port. Other input ports, *i0* and *i1* in figure 3.7, are intended to describe data streams which might be important for end-to-end analysis. The output ports, *o* and *o0* in figure 3.7, carry the output event model derived from task activation and scheduling. Each output port carries the same output event model because the task can be activated only by the single input port *a*. Several tasks on an ECU might be activated synchronously. Therefor, the activation sources have to be synchronized, clarified by a little clock within the source symbol, like illustrated by figure 3.12. Thus, it is possible to realize activation offsets between tasks on an ECU.
The OSEK OS scheduling policy described in section 3.2.2 is similar to that of RMS. In addition, the **blocking time due to the cooperative scheduling of processes** has to be considered as well as the **activation offsets**. So, the feasibility check for RMS (see [19, p. 90]) has to be extended to consider this blocking time, see [19, p. 114]. If all activation offsets are zero, it is trivial to find the critical instant for each task. As real-world systems often use activation offsets in order to reduce task preemptions by higher priority tasks, finding the critical instant is not trivial. The scheduling analysis has to examine each possible point within the hyperperiod of the schedule, i.e. the time range about the least common multiple of the task periods. So, possible points are task respectively process activations and terminations. SymTA/S reduces this mathematical complexity by sophis-



www.manaraa.com

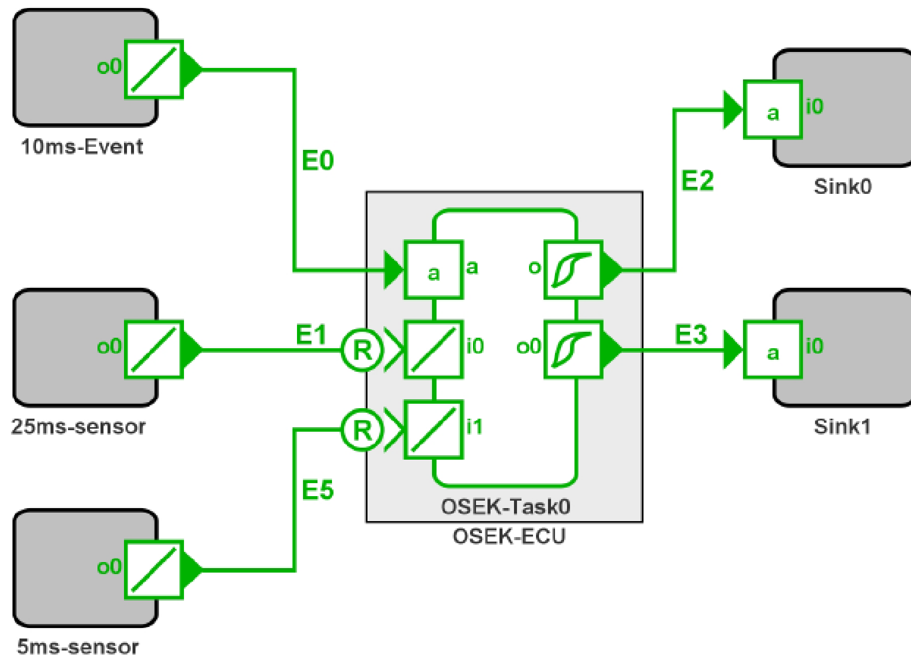



Figure 3.7: SymTA/S OSEK-task

ticated algorithms [44, p. 29]. These algorithms are intellectual property of Symta Vision. In addition to this complexity reduction, SymTA/S provides **three analysis accuracy levels**. The most accurate full offset analysis examines all possible worst-case candidates but takes most time. The medium accurate approximate offset analysis reduces the number of possible candidates conservatively whereas the least accurate offset blind analysis ignores the offsets. This leads to a pessimistic result but requires not much time. In general, analysis time could grow linear with the number of worst-case candidates. The number of candidates could also be increased using asynchronous schedules in the same system [44, p. 30].
Further analysis refinements are possible by specifying **process periods and offsets** of OSEK OS tasks. Period means that the process is executed only every e.g. second or third time the corresponding task is executed. Given a period of two, an offset of one means that the process is executed the second time, next the fourth time, the sixth, and so on.

### 3.3.4 CAN Scheduling

A CAN-bus is a broadcast bus where several processors are connected by a common interface. Figure 3.8 illustrates the architecture of a CAN-bus. A CAN-frame can carry between one and eight bytes user data. Each frame has a unique 11-bit ID that also determines the priority of the frame. This ID identifies the sender of the frame as well. According to this ID, a station on the CAN-bus decides if it has to receive the frame that



المنارة للاستشارات
www.manaraa.com

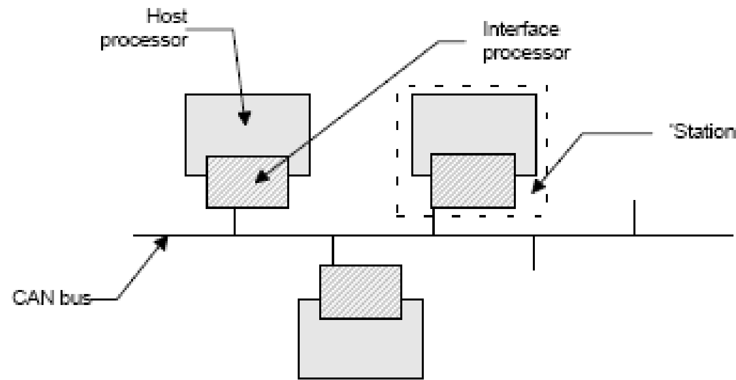



Figure 3.8: CAN-bus architecture, from [13, p. 1]

is currently sent. The ID is also used for collision resolution via a carrier sense multiple access collision resolution protocol. For detailed information it is recommend to read paper [13]. Frame transmission is non-preemptive, i.e. a high priority frame can be blocked by a low priority one.
In order to create a CAN-bus within SymTA/S, a so-called *COM-bus* has to be created. One or more frames can be added and mapped to the COM-bus. For each frame, the priority and packet size has to be specified. Besides some overhead, only the given packet size and bus speed are crucial for determining the transmission time. It is important to choose the bus speed according to the used execution time unit. SymTA/S automatically takes the scheduling overhead due to the access protocol into account. However, re-transmission due to lost or damaged frames and sent frame requests, like described in [13, p. 4 cpp.], are not considered by SymTA/S. It also assumes that data is written to the interface processor at the end of a task. Within a CAN frame, a so-called stuffing bit is inserted after four identical bits. These additional bits lengthen transmission time of a frame. SymTA/S allows making an assumption about these stuffing bits: inserting as much as possible, half of the possible and no stuffing bits.
A CAN frame can be sent periodic, sporadic or on demand. Therefor, SymTA/S provides **CAN-frame transmission modes** periodic, direct, mixed and none. Figure 3.9 illustrates a CAN-frame with mixed transmission mode which is mapped to a CAN-bus. It is transmitted periodically with a period of 10. Therefor, an external source is connected to the COM-port *a* of the frame. Additional, it is transmitted if an event occurs on port *i0*. In the example, it is set to simple periodic with a period of 15. The remaining port *i1* is connected to another external source which could be a task writing data into the buffer of its CAN-interface. In contrast to an event occurring on *i0*, the arrival of this data does not trigger a frame transmission. Therefor, the data has to be buffered which is illustrated by the little *R* at port *i1*. This is an example for an event model interface, like described in section 3.3.2. A frame with transmission mode none is ignored during analysis. Each port pair except *a* and *o* correspondents to a so-called signal. In principle, a signal contains a



المنارة للاستشارات
www.manaraa.com

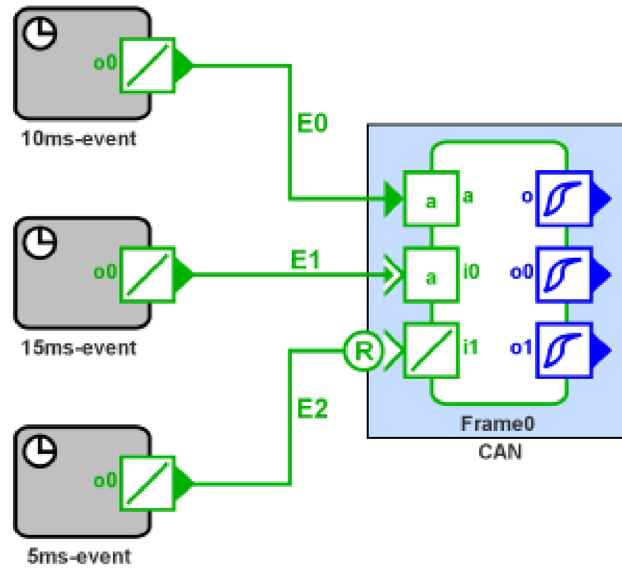



Figure 3.9: SymTA/S CAN-frame

message for a CAN-bus station. A CAN-frame is like an envelope that transports several messages. The first port pair *a* and *o* is for frame-activation only. *o* is used to propagate the output event model derived from COM-port activation. The other port pairs consider signal-related event models. So, *o0* has an output event model of $T = 15$ plus some jitter because the signal is a so-called triggered one that causes a frame transmission. The output port *o1* has $T = 10$ plus some jitter because it is a so-called pending signal, i.e. it does not trigger a frame activation and therefor its output event model is derived from that of *o*. So, ports can be added only pair wise. The size and position of signals within a frame are ignored during analysis. It is important to check if a signal that is thought as pending signal is really set to pending within SymTA/S, even if the transmission mode of the frame is set to periodic. The symbol left of the import port indicates the transfer property of the incoming signal. So, the signal on port *i0* shown by figure 3.9 is triggered respectively it is responsible for frame activation. This is clarified by the little *a*. In contrast, the signal on input port *i1* is pending, clarified on the symbol of its incoming event mode.
CAN-frames that have a sporadic activating input event model describe a kind of dynamic load. Besides assuming the activation of all possible frames, SymTA/S also allows assuming only the activation of periodic frames or bounding the number of asynchronous activations to a certain value. So, analysis can be refined to get closer to the real-world. As described in section 3.3.3, SymTA/S also provides three analysis accuracy levels for CAN-scheduling.



المنارة للاستشارات
www.manaraa.com

## 3.3.5 End-to-End Analysis

### Preface

In order to avoid too pessimistic WCRTs, SymTA/S has to take timing dependencies and especially synchronization dependencies along system paths into account. So, end-to-end analysis has to consider local scheduling analyses and vice versa. In this section, the difference between the described analysis accuracy levels becomes more apparent. SymTA/S distinguishes between signal-path and event-triggered path analysis.

### Event-Triggered Path Analysis

The event-triggered path analysis examines the cases where the output event of an application element is the activating input event of the subsequent one. The application elements are synchronized to each other in that way that a termination immediately causes activation. In figure 3.5, the termination of *OSEK-Task0* on *OSEK-ECU0* causes the immediate activation of *CAN-Frame0*. The transmitted CAN-frame activates *OSEK-Task2* on *OSEK-ECU1*. After its execution, *CAN-Frame1* is triggered which finally activates *OSEK-Task1* on *OSEK-ECU0*. The activations are not buffered and so the activation of *OSEK-Task1* directly depends on the activation time of *OSEK-Task0*. So, the application elements are synchronized implicitly.
When choosing *offset blind analysis*, SymTA/S assumes a simultaneous activation of the tasks respectively CAN-frames in order to determine the WCRTs. The Gantt-chart in figure 3.10 illustrates this. Possible offsets due to the implicit synchronization along the event triggered path are not considered. The end-to-end WCRT is composed of these pessimistic local WCRTs.
When choosing *full offset analysis*, the results are less pessimistic, illustrated by figure 3.11. Under consideration of the given timing dependencies, SymTA/S recognizes that *CAN-Frame1* is never blocked by *CAN-Frame0*. Further, the activation of *OSEK-Task1* is correctly not simultaneous with *OSEK-Task0*. Again, the end-to-end WCRT is composed of the local WCRTs.

### Signal-Path Analysis with CAN-communication

Signal-path analysis catches the cases where communication is buffered like that is a common case in automotive industry. Such paths through the system usually do not carry activation events. Instead, they are carrying data, e.g. sensor data. In order to describe this issue, an extended version of the system described above is used. It is illustrated by figure 3.12. The communication per **CAN** is buffered, i.e. the termination of *OSEK-Task0* does not activate the *CAN-Frame0*. Instead, the data is written to a register. *CAN-Frame0* is activated periodically by *Source1* with a period of 10 and reads the register every time it is activated. After transmission, the data is written to a register that is read by *OSEK-Task2* which is activated periodically by *Source2* with a period of 10. After termination of this



المنارة للاستشارات
www.manaraa.com

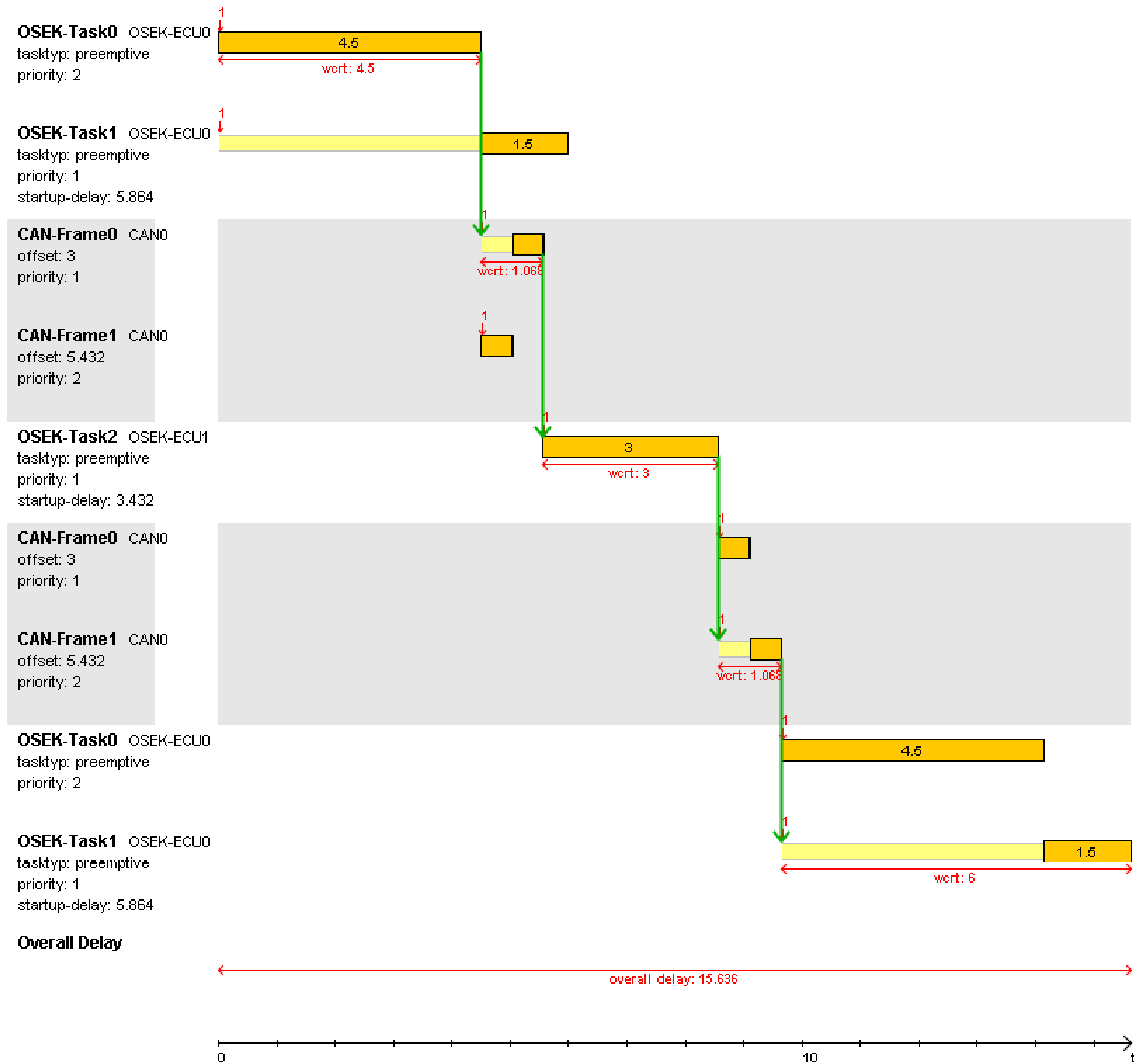



Figure 3.10: SymTA/S offset blind analysis Gantt-chart



المنارة للاستشارات
www.manaraa.com

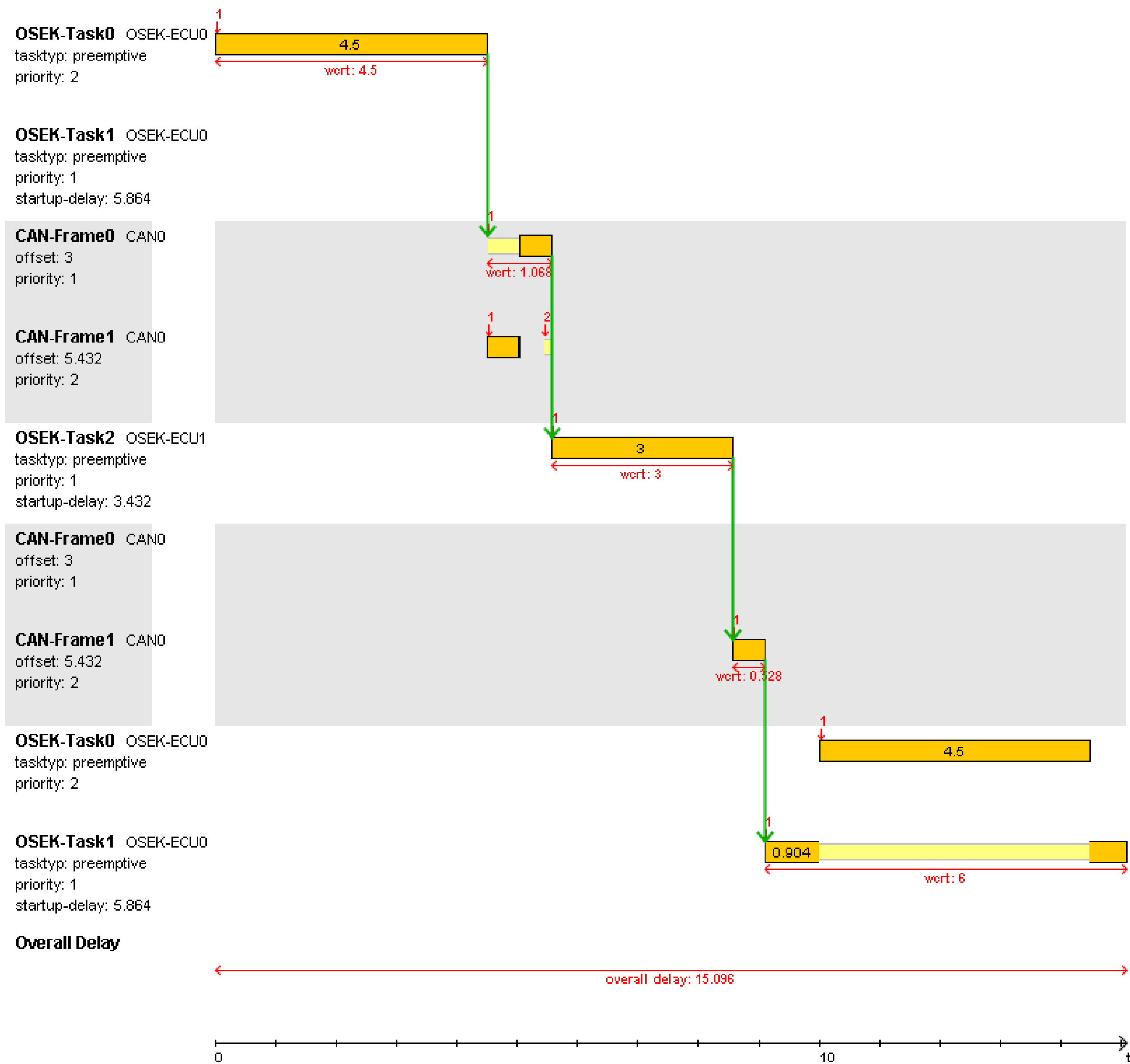



Figure 3.11: SymTA/S full offset analysis Gantt-chart



المنارة للاستشارات
www.manaraa.com

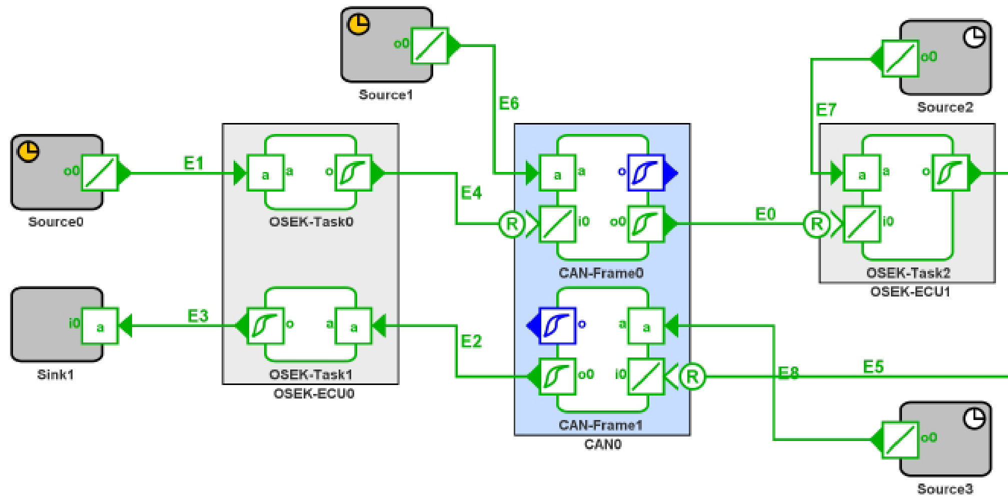



Figure 3.12: SymTA/S example system with buffered CAN-communication

task, the data is written to a register again. This register is read by *CAN-Frame1* which is activated periodically by *Source3* with a period of 10. After transmission, *CAN-Frame1* activates *OSEK-Task2* immediately. *Source0* and *Source1* as well as *Source2* and *Source3* are synchronized. The little clock within the sources clarifies that. The events triggered by the synchronized sources have an offset. The activation of *CAN-Frame0* has an offset of 5 to the activation of *OSEK-Task0*. So, the data written by *OSEK-Task0* just waits a time span of 0.5 before it is transmitted by the frame when assuming a WCET of 4.5 for *OSEK-Task0*. This is done similarly for *Source2* and *Source3* on *OSEK-ECU1*. However, the "clocks" of *OSEK-ECU0* and *ECU-ECU1* are not synchronized. So, SymTA/S has to assume a delay of 10 on *E0* which is the reading period of the register *CAN-Frame0* writes to. The reason is that the analysis leaves a so-called *synchronized group* of application elements. Synchronized groups are activation elements that are activated by synchronized sources or which are synchronized implicitly (described in the antecedent section). This delay is illustrated by figure 3.13. Analysis is performed with full offset method. The buffered communication has further effects on analysis. *CAN-Frame0* and *CAN-Frame1* as well as *OSEK-ECU0* and *OSEK-ECU1* are not synchronized any more, whether implicitly nor explicitly. So, SymTA/S assumes simultaneous activation of the task respectively frames in order to determine the WCRTs.

It is important to note that using buffered communication and different synchronization groups can lead to numerous under- and over-sampling effects, i.e. data is written more often than it is read and vice versa. **Under-sampling** is easy to handle because the last read data is always up-to-date, i.e. one data instance can be read one time at most. In this case, SymTA/S assumes a maximum waiting time between two synchronized groups which contributes to the global (path) WCRT. This maximum waiting time is the maximum distance between two writing events, e.g. the writing cycle time plus jitter. This



المنارة للاستشارات
www.manaraa.com

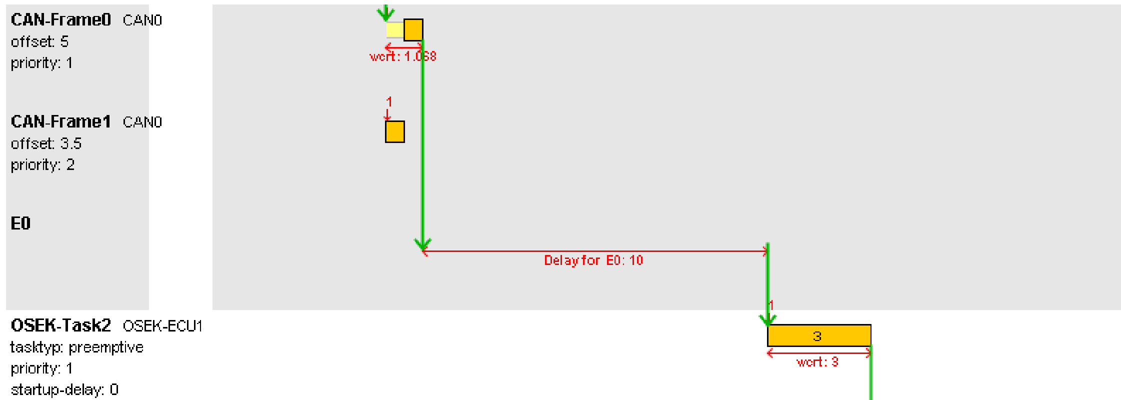



Figure 3.13: SymTA/S buffered CAN-communication Gantt-chart extract

results from the examination of a certain data instance. If a higher delay would be assumed, e.g. because the reading cycle time is higher, another data instance would be read and not the examined one. So, the maximum distance between two writing events is the longest time a certain data instance can wait before it is overwritten.
On points of **over-sampling**, a data instance could be read multiple times. So, SymTA/S provides two analysis semantics: First Through and Maximum Age. The *First Through* semantic determines the WCRT which the data needs to traverse the end-to-end path the first time. The maximum waiting time between two synchronized groups is the maximum distance between two reading events (e.g. period time plus jitter), see [44, p. 34]. Imagine the above described example with buffered communication in figure 3.12. However, the periods of *Source0* and *Source1* are set to 25. So, over-sampling occurs on *E0* respectively *OSEK-Task2*. Using First Through semantic, the delay on *E0* is 10 because *CAN-Frame0* writes the register every 25 time units, plus some jitter of frame transmission, but *OSEK-Task2* reads the register with a cycle time of 10 without jitter. So, the data waits at most 10 time units in the register before it is read the first time.
SymTA/S catches much more situations that could occur due to over- and under-sampling effects. A description would go beyond the scope of this work. Because over-sampling allows reading the same data multiple times, it might be of interest when the last instance of the data has traversed the path. The *Maximum Age* semantic determines the WCRT of the data which is read last at the over-sampling point. The next read access at the over-sampling point would read another data instance. In the example which is illustrated by figure 3.12 and where the periods of *Source0* and *Source1* are set to 25, a maximum waiting time of 25.636 is determined on *E0*. That is the maximum distance between two arriving CAN-frames [44, p. 34] which consists of the sender cycle time plus the sender sided jitter. In the worst case, the data has to wait this time span in the register until it is overwritten with new data.
In some cases it is necessary to know how many instances of the same data traverse the path respectively reach the end. Imagine unsynchronized over-sampling on a register that



المنارة للاستشارات
www.manaraa.com

is written with a period of 25 and read with a period of 10. So, the same data can be read 2 or 3 times within a period of 25 if writing takes less than 5 time units. The maximum number of read data instances might be derived from the difference of the BCRTs using First Through semantic and Maximum Age semantic. For example, the difference between both analyses is assumed to be 22 time units and the reading event occurs every 10 time units. So, maximal 3 data instances could traverse the path. For example, the first at t = 0 relative to the BCRT when using First Through semantic, the second at t = 10 relative to this BCRT and the third one at t = 20 relative to this BCRT. The BCRT has to be taken for both analyses because it is assumed that both analyses start from the same situation where the data is read immediately after writing. Unfortunately, the BCRT analysis does not work properly. For example, the sub-path from *OSEK-Task0* to *CAN-Frame0* in the system illustrated by figure 3.12 always contributes 5 time units to the path WCRT because the *CAN-Frame0* is activated with an offset of 5 to the task and the task's WCET is below 5 time units. During BCET analysis, this 5 time units are not taken into account. When removing the synchronization of *Source0* and *Source1*, the sub-path from *OSEK-Task0* to *CAN-Frame0* correctly contributes 3 to the path BCRT because SymTA/S assumes the BCET of *OSEK-Task0* which is about 3 and an immediate reading event of *CAN-Frame0*. This BCRT analysis problem is fixed in higher versions of SymTA/S. Sampling effects due to jitter also have to be taken into account. They are described later in section 3.5.
The SymTA/S user should be able to understand and to interpret the results displayed because of the numerous effects that could occur. Especially, it is necessary to understand why SymTA/S assumes a certain blocking time in a certain situation. This gives additional information about the correct system modeling within SymTA/S which has to be done very carefully.

### Signal-Path Analysis with FlexRay-communication

SymTA/S also allows scheduling analysis using FlexRay-communication. FlexRay provides deterministic frame transmission following the *Time Division Multiple Access* approach. Each frame has a fixed (time) slot which is used for transmission. In principle, only the bus configuration is needed for analysis, i.e. time slot length, which frame is assigned to which time slot, etc. SymTA/S provides the import of the standard FlexRay exchange format.
Imagine the example system illustrated by figure 3.12 with a FlexRay- instead of CAN-bus with a cycle time of 10. If the FlexRay-bus and both ECUs are not synchronized, SymTA/S assumes a delay of 10 on each event-stream between ECU and bus like illustrated by figure 3.14. Again, implicit synchronization that might lead to less pessimistic delays is not considered by SymTA/S. Path analysis with CAN-communication would lead to the same results if the CAN-frames are asynchronous to the sending task respectively ECU.
FlexRay allows the synchronization of the bus with *all* connected ECUs. Such full synchronized systems might have short response times but are less flexible, e.g. when in-



المنارة للاستشارات
www.manaraa.com

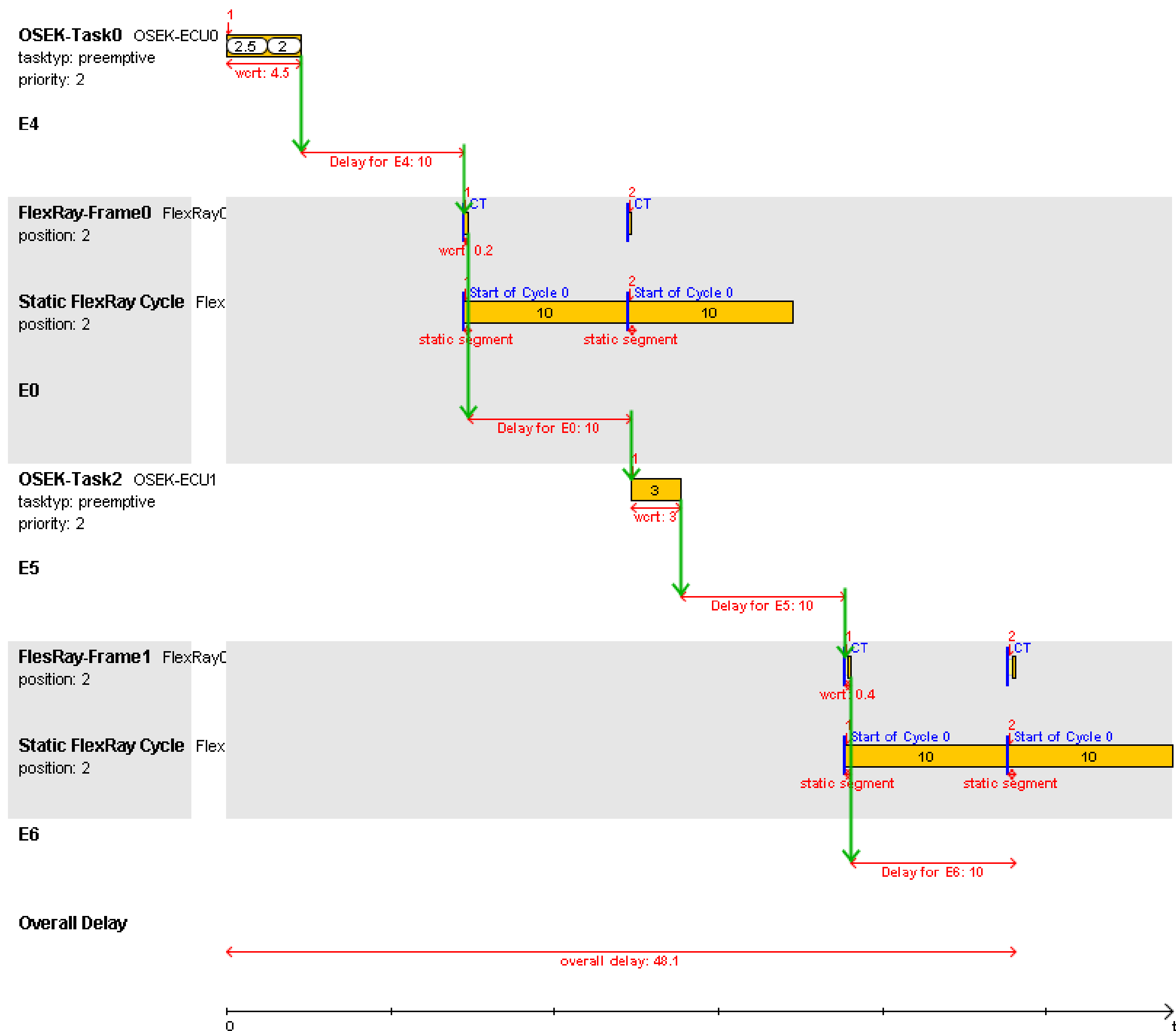



Figure 3.14: SymTA/S asynchronous FlexRay-communication Gantt-chart



المنارة للاستشارات
www.manaraa.com

serting additional code. Figure 3.15 illustrates the path WCRT using a full synchronized configuration of the example system that is described above (compare with figure 3.14). Obviously, the delays between task termination, frame transmission and frame reading are reduced and sampling effects due to jitter does not occur any more. However, if the WCRT of the 10ms-task on *OSEK-ECU0* increases, the synchronization might have to be changed. The synchronization configuration is determined by the design space exploration of SymTA/S (see section 3.3.7). FlexRay-scheduling is not examined further in scope of this work because the examples described later use CAN-scheduling. However, the example shows that FlexRay-analysis in combination with the design space exploration functionality might be useful for the development of such full synchronized systems.

### 3.3.6 Sensitivity Analysis Plug-In

The SymTA/S sensitivity analysis plug-in determines the sensitivity of the system with respect to a certain system parameter. This functionality can be distributed to several computers. On the one hand, sensitivity analysis checks how a parameter can be changed without overloading the processor or violating a timing constraint. So, the plug-in can be used to determine a safety margin within an existing system. On the other hand, it is determined how certain parameter has to be changed in order to fulfil a timing constraint. It is important to note that it is not sufficient to check only the resource speed sensitivity. It is necessary to the check each task's WCET because utilization based schedulability tests do not guarantee meeting all timing constraints, see section 3.4.3. In case of a not working system, it can help determining which parameter to change in order to make the system working.
In particular, SymTA/S provides the sensitivity analysis of the WCET of tasks, processes, bus messages, of the resource speed and of the source period and jitter. The investigated parameter is varied and it is checked if the systems still fulfils all timing constraints, e.g. an end-to-end-path constraint. The appropriate value is searched by a binary search [45, p. 16]. The search interval depends on the fulfilment of all system constraints and on predefined values like maximum resource utilization. The sensitivity analysis considers the variation of only one parameter per timing constraint. For example, the analysis result of this parameter would lead to the predefined maximum processor utilization. Figure 3.16 shows the sensitivity output of the resource speed analysis. This analysis determines the spare respectively necessary processor speed which allows meeting all system constraints. The described analysis is a so-called one-dimensional sensitivity analysis. SymTA/S also provides so-called dependency diagrams. They illustrate the effect of a parameter (jitter or offset) on an application element's WCRT. Figure 3.17 illustrates the influence of *Source0*'s (period: 25) jitter on the WCRT of *OSEK-Task0* (WCET: 4.5) from the example system illustrated by figure 3.12. It easy to see that large source jitter, i.e. delayed task activation, affects the WCRT of the subsequent task activation.



المنارة للاستشارات
www.manaraa.com

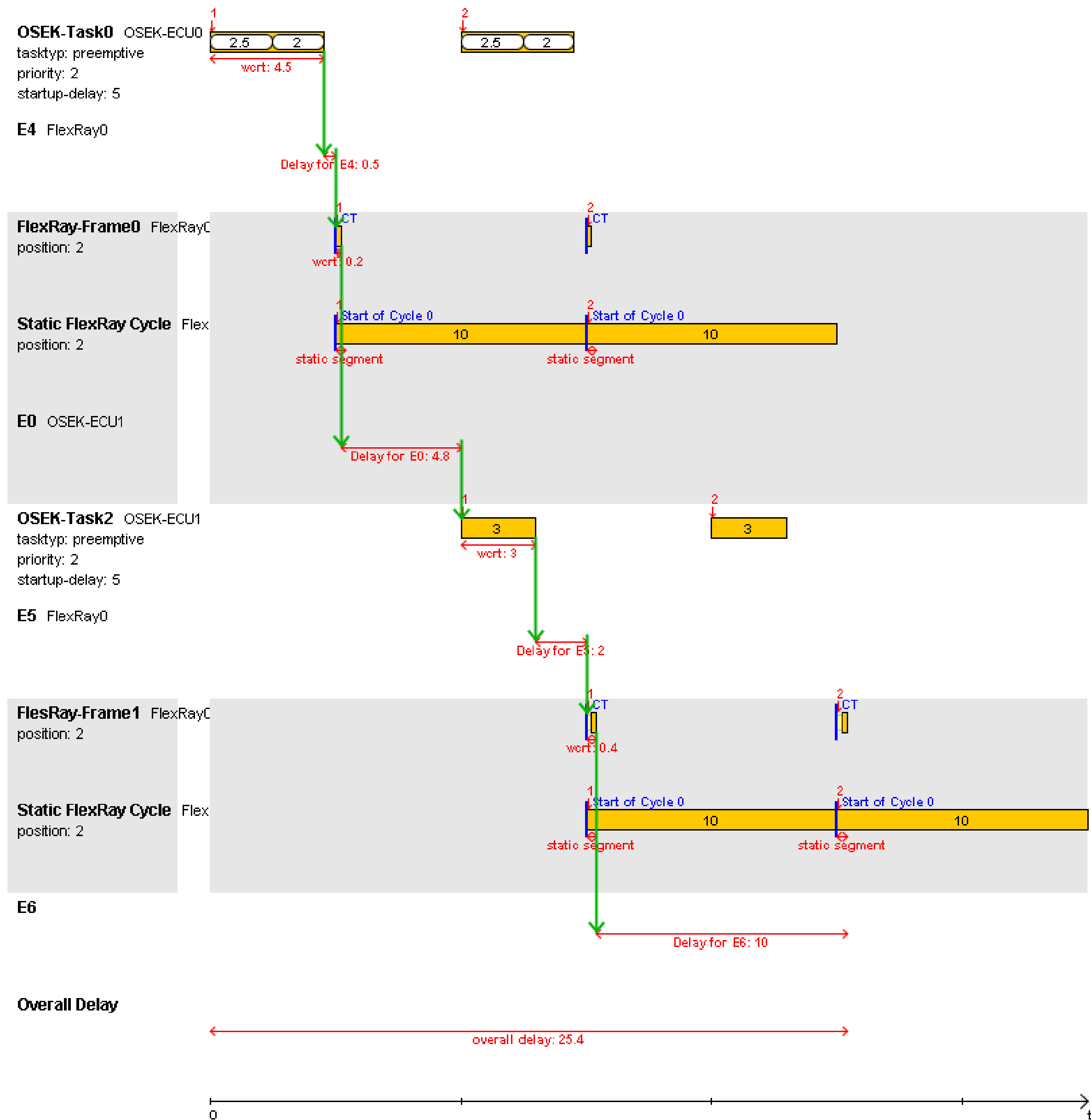



Figure 3.15: SymTA/S synchronous FlexRay-communication Gantt-chart



المنارة للاستشارات
www.manaraa.com

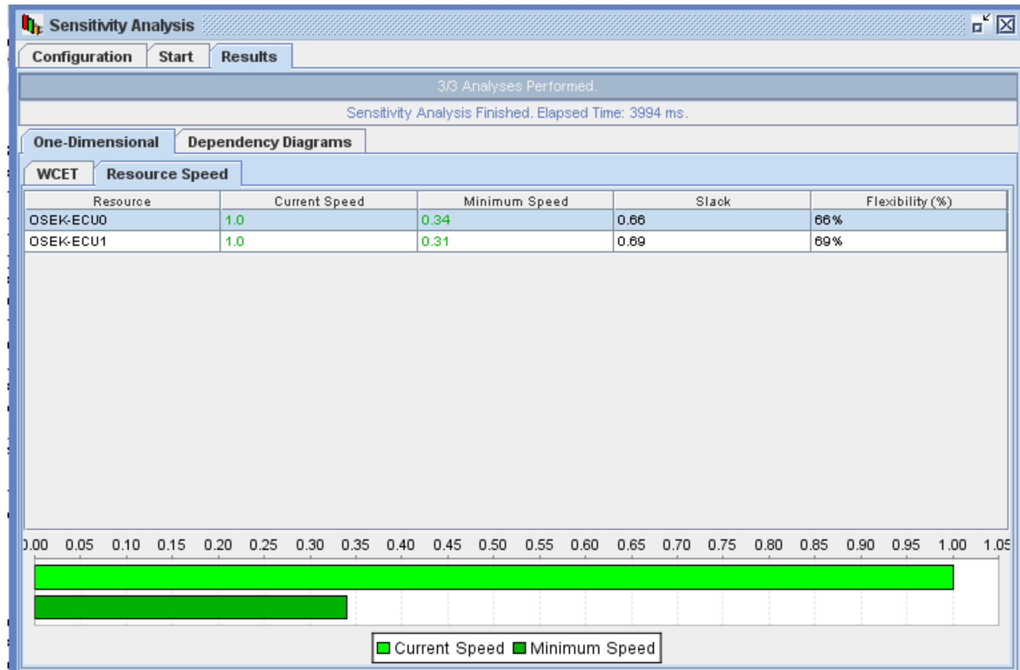



Figure 3.16: SymTA/S resource speed sensitivity analysis

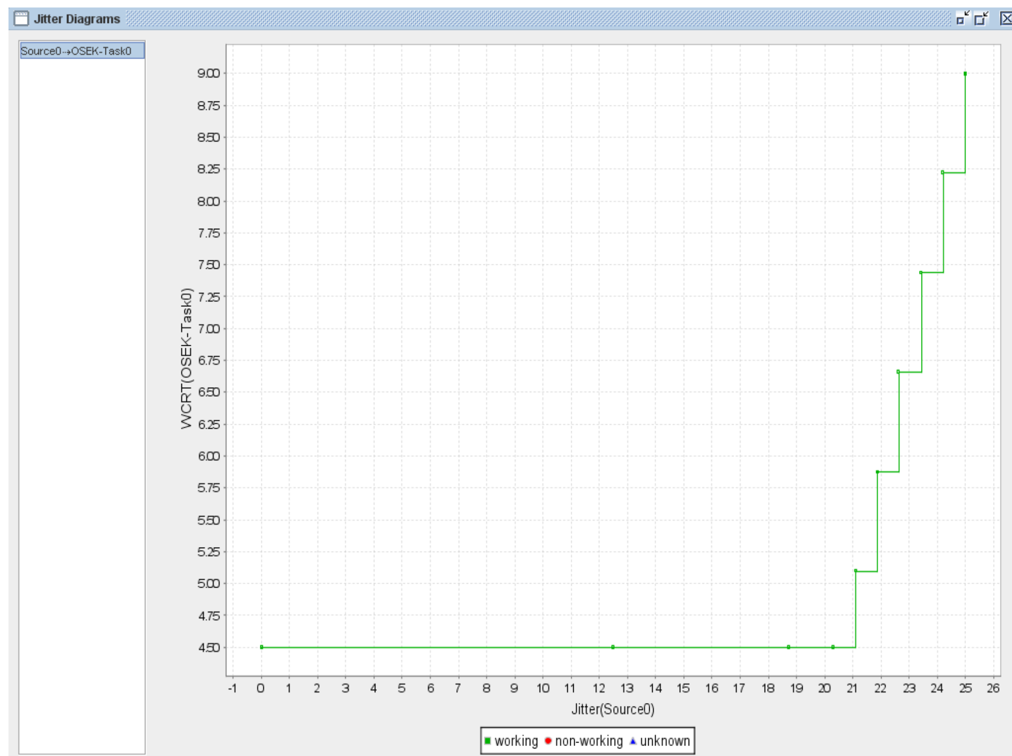



Figure 3.17: SymTA/S jitter dependency diagram



المنارة للاستشارات
www.manaraa.com

### 3.3.7 Design Space Exploration Plug-In

The SymTA/S design space exploration plug-in allows the optimization of priorities on resources like busses or ECUs and of offsets on synchronized sources. Like sensitivity analysis, the execution of this function can be distributed to several computers. After choosing the search space, i.e. the parameters that will be optimized, the optimization objectives have to be selected. SymTA/s provides the optimization of individual system path or jitter constraints. Further, global optimization objectives like minimization of average path latency, minimization of parameter changes or the optimization of path and jitter constraints in general can be selected. In cases of path and jitter optimization, a penalty for constraint misses can be set in order to guide the optimization in a certain direction. The so-called SymTA/S design space exploration loop is illustrated by figure 3.18. In terms

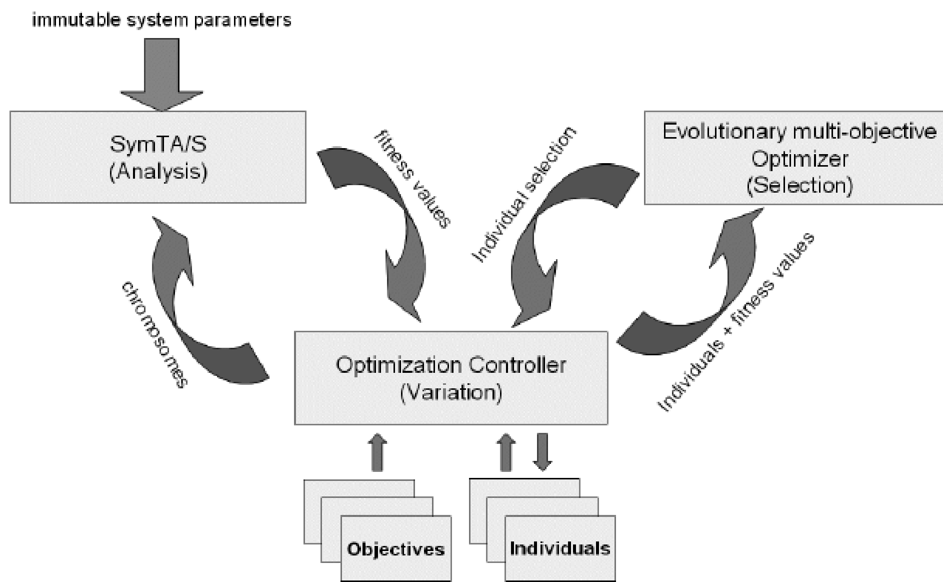



Figure 3.18: SymTA/S design space exploration loop

of SymTA/S, a chromosome is a modifiable system parameter. An individual is a certain system configuration which consists of an unmodifiable part and chromosomes. During optimization, SymTA/S analyzes several individuals. Therefor, it is initialized with the unmodifiable system parameters. For each individual to be analyzed, its chromosomes are applied to SymTA/S. Depending on the analysis results, the *optimization controller* creates the fitness values of the individuals according to the optimization objectives. The individuals plus their fitness vectors are submitted to the *evolutionary multi-objective optimizer*. Considering the fitness vectors, it divides the individuals into two sets. Therefor, the FEMO (Fair Evolutionary Multiobjective Optimizer), NSGA2 (Nondominated Sorting Genetic Algorithm) or, as default, SPEA2 (Strength Pareto Evolutionary Algorithm) are used. These sets are submitted to the *optimization controller* again. One set contains those individuals to be deleted (eliminated). The other one contains those individuals on



المنارة للاستشارات
www.manaraa.com

which mutation and cross-over operations are performed in order to create a new population. Then, the optimization cycle starts again. Optimization output is one or more pareto-optimal individuals. I.e. it is not possible to improve the value of a parameter without degrading the value of another parameter within the configuration. This holds for all pareto-optimal individuals. It is very important to note that the optimal individual is not guaranteed to be found. The reason is that mutation and crossover operations depend on probabilities. So, a random number generator is initialized with a seed. In order to get reproducible results, this seed has to be noticed. Further optimization parameters like (initial) population size, number of generations and the mutation and crossover probability could be set. However, the effect of these parameters is not investigated in detail in scope of this work. Several optimization runs concerning the activation offsets of an example system show that even using more and bigger populations just increases analysis time on the test system but does not provide much better results in general. So, default values are used in the following for design space exploration. Even manually generated configurations might be sufficient and might be gained more quickly as long as the systems to be analyzed are simple enough.

## 3.4 Scenario 1: Analysis of an Existing System

### 3.4.1 Preface

The first scenario to be examined concerns the analysis of an existing system that is already used for WCET analysis, see section 2.6.1. That is an ECU with Infineon TriCore 1766 processor running a certain OSEK OS-implementation and several tasks with periods of amongst others 10 ms, 20 ms, 30 ms, 50 ms and 100 ms. The task priorities increase with decreasing period time, i.e. the fastest has the highest priority while the 100ms-task has the lowest one. The tasks are divided into processes. The second fastest task is realized as processes within the fastest task. They are executed every second time the fastest task runs. This is taken into account by SymTA/S through process offset and period like described in section 3.3.3. All tasks with a priority equal or lower than the 10ms-task can preempt each other only at process boundaries. This is modeled by setting their *task type* to cooperative and assigning a shared resource to them within SymTA/S. All faster tasks can preempt these tasks anytime. ISRs can preempt all tasks anytime. The task activations are synchronized and they have certain activation offsets.
It is intended to determine the execution times of several tasks respectively processes which might lead to a potential scheduling problem when appearing in a certain (worst-case) combination which shall be determined by SymTA/S. The system analysis is performed by using both common execution time measurement methods that are described in section 2.6.2. For precise scheduling analysis, the execution times of all processes are required because a task might be blocked by a lower priority task due to cooperative scheduling. Section 2.6.2 describes that the process execution times that are extracted



المنارة للاستشارات
www.manaraa.com

from hardware-traces might lead to different task execution times than the task execution times extracted from the same raw data. These effect on scheduling analysis is examined in section 3.4.2. Section 3.4.3 describes scheduling analysis with process execution times derived from INCA.
The used time unit is $\mu$s ($10^{-6}$s) because SymTA/S provides only four decimal places which were not sufficient when using ms ($10^{-3}$s) due to the short process execution times. The *activation buffer* is set to one because only one instance of a certain task is allowed to be activated. So, the task's period time is applied to each task as timing constraint.

### 3.4.2 Analysis using Execution Times derived from Hardware-Traces

#### Analysis Using Task Execution Times

The following scheduling analysis bases on task execution times extracted from hardware-traces like described in section 2.6.2. The debugger records only 850 ms, so that the execution time measurement interval is very short. The ISRs that are extracted from the hardware-trace are taken into account by determining the average period of each ISR. The simple sporadic event model is assumed. The ISR activation is not synchronized and they are assumed to be scheduled non-preemptive. The OSEK-kernel priority is set higher than the task priorities but lower than the interrupt priorities, according to figure 3.1. The tasks are set to preemptive scheduling because setting them to cooperative scheduling would cause too pessimistic blocking times of high priority tasks.
Figure 3.19 illustrates the used SymTA/S model. It comprises 7 tasks (upper row of application elements) and 13 ISRs (lower rows of application elements) and their associated activation sources. The model is refined by assuming an overhead of 0.92 $\mu$s per task termination. The overhead caused by task activation could not be extracted from the hardware-trace.
As first **scheduling analysis** result, the maximum processor utilization is 61.9%. Figure 3.20, which is extracted from the analysis report file *RPF00*, shows that each task meets its timing constraint, i.e. the WCRTs do not exhaust the period times. The report files and SymTA/S system files of all analyses are included by the enclosed archive. The synchronized sources within the system and hence the task activation offsets are not listed. This is fixed in newer versions of SymTA/S.
The **sensitivity analysis** individually checks the task WCETs, ISR activation periods and their jitter. The maximum respectively minimum values are determined under which the maximum processor utilizations of 97% (adjustable value) is reached respectively all timing constraints are met. So, sensitivity analysis for each task WCET and ISR source jitter might be required to estimate a safety margin for the system. In the current example, all task WCETs have a tolerance of at least 139% (see the enclosed report file *RPF00*), i.e. at least 139% of the WCET of each task can be added to its current WCET. The system would still meet all timing constraints under that circumstances.



المنارة للاستشارات
www.manaraa.com

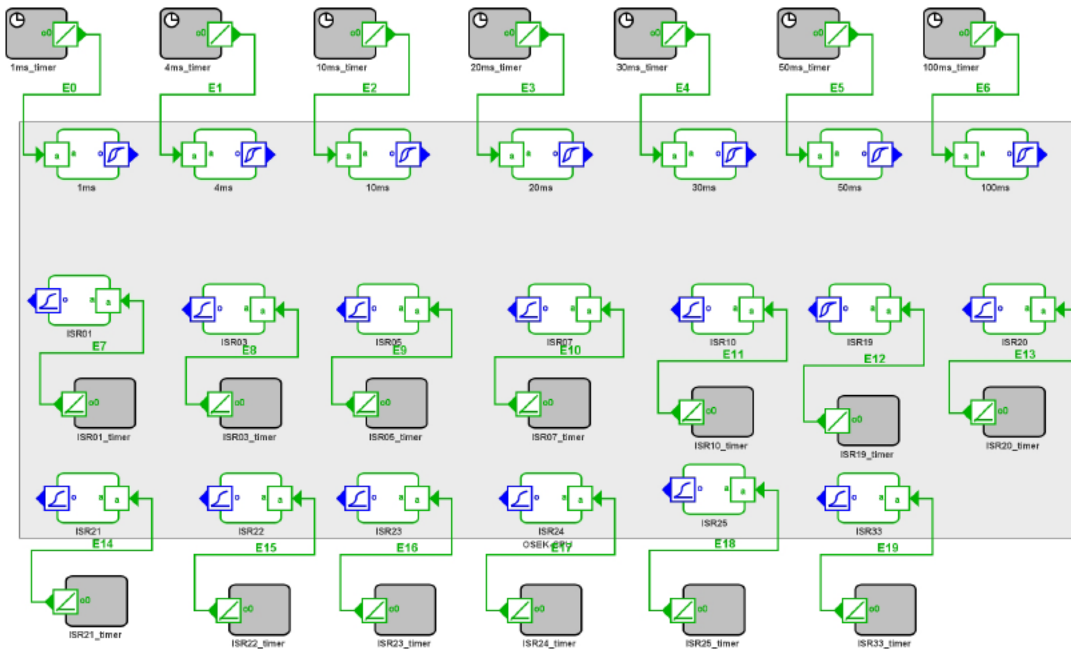



Figure 3.19: SymTA/S scenario 1: system model

**Observed Paths**

| Path | # Events | Latency | Constraint | Buffer | Status |
|---|---|---|---|---|---|
| 1ms_timer;o01 ms;o | 1 | [8.86, 257.82] | 1000 | 1 | SUCCESS |
| 4ms_timer;o04 ms;o | 1 | [91, 476.04] | 4000 | 1 | SUCCESS |
| 10ms_timer;o0 10ms;o | 1 | [93.82, 3801.52] | 10000 | 1 | SUCCESS |
| 20ms_timer;o0 20ms;o | 1 | [86.36, 3217.42] | 20000 | 1 | SUCCESS |
| 30ms_timer;o0 30ms;o | 1 | [909.72, 2469.48] | 30000 | 1 | SUCCESS |
| 50ms_timer;o0 50ms;o | 1 | [66.04, 1579.6] | 50000 | 1 | SUCCESS |
| 100ms_timer;o 0100ms;o | 1 | [146.48, 3873.18] | 100000 | 1 | SUCCESS |

Figure 3.20: SymTA/S scenario 1: observed paths using task execution times derived from hardware-traces



المنارة للاستشارات
www.manaraa.com

In principle, the **design space exploration** functionality is not required for this use scenario because no problems are detected. Nevertheless, it could be used to check possible system configuration optimizations that lead to shorter WCRTs. An exemplary optimization run starts with the initial activation offsets. As search space, the activation offsets of the synchronized sources on *OSEK-ECU* are specified. The smallest modification unit is set to 100 because SymTA/S does not provide a higher one. However, for the current example a smallest unit of 1000 is required because the activation offsets of the real system are adjustable only in units of 1 ms. So, the activation offsets determined by SymTA/S might be rounded arbitrary to multiples of 1000 for subsequent scheduling analysis in order to represent a real configuration of the system. Another possibility is to use a suitable time unit within SymTA/S, e.g. of at least 10 $\mu$s. This would reduce precision, especially when dealing with very short execution times, because SymTA/S provides only four decimal places. For design space exploration, all task WCRTs and the minimization of parameter changes are specified as optimization objectives. Further settings are default (seed: 1201090819042). After approximately 90 minutes, SymTA/S determines no better configuration than the initial configuration, running on the test system. On a more powerful system, this analysis is assumed to run faster.

### Analysis Using Process Execution Times

The second scheduling analysis uses the same system model and settings like the analysis described in the antecedent section. Now, process execution times are used which are derived from the same hardware-trace. The processes of some tasks are scheduled cooperatively like in the real system. It is intended to check the effect of the not-considered overhead (described in section 2.6.2) on the analysis. Currently, the hardware-trace processing scripts do not consider all overhead on process-level.
As result, all tasks meet their timing constraints and the WCRTs are much lower than the ones based on task execution times, see figure 3.21 and report file *RPF01*. It is obvious that the not considered overhead is not insignificant. The maximum processor utilization of 54.9% is lower than that basing on task execution times. Obviously, the result based on that input is unsafe and is not suitable for system verification. An adaptation of the hardware-trace processing scripts is necessary when holding on this approach. Detailed analysis results can be taken from the report file *RPF01*.
**Sensitivity analysis** on the process WCETs, ISR activation periods and jitter takes over 26 hours on the test system. This might be caused by the big search interval for the binary search and the numerous processes that are analyzed. So, a lot of configurations have to be tested. That lengthens analysis time. SymTA/S determines the maximum possible WCET of each process and the maximum ISR activation period and jitter, see report file *RPF01*. The WCET tolerance is very high. As analysis settings, the maximum processor utilization is set to 97% and the task timing constraints are their periods. **Design space exploration** takes approximately 90 minutes on the test system. The same settings are used as described in the antecedent section. Again, SymTA/S determines no better system



المنارة للاستشارات
www.manaraa.com

**Observed Paths**

| Path | # Events | Latency | Constraint | Buffer | Status |
|---|---|---|---|---|---|
| 1ms_timer;o01ms;o | 1 | [24.46, 202.72] | 1000 | 1 | SUCCESS |
| 4ms_timer;o04ms;o | 1 | [73.66, 402.14] | 4000 | 1 | SUCCESS |
| 10ms_timer;o010ms;o | 1 | [2303.78, 3724.31] | 10000 | 1 | SUCCESS |
| 20ms_timer;o020ms;o | 1 | [185.32, 3139.41] | 20000 | 1 | SUCCESS |
| 30ms_timer;o030ms;o | 1 | [889.54, 2314.19] | 30000 | 1 | SUCCESS |
| 50ms_timer;o050ms;o | 1 | [33.98, 1383.17] | 50000 | 1 | SUCCESS |
| 100ms_timer;o0100ms;o | 1 | [147.46, 3432.83] | 100000 | 1 | SUCCESS |

Figure 3.21: SymTA/S scenario 1: observed paths using process execution times derived from hardware-traces

configuration for the current system.

### 3.4.3 Analysis using Execution Times measured by INCA

#### Scheduling Analysis

The execution times for the following analysis are measured by INCA while running a SoftCar RT-script which is described in section 2.6.2. In order to provide a very precise scheduling analysis, the process execution times are measured. As scheduling analysis result, all tasks meet their timing constraints. This illustrated by figure 3.22 from report file *RPF02*. SymTA/S determines a highest possible processor utilization of 97.64%. The highest utilization measured by INCA is about 72%. This might lead to the assumption that this scheduling analysis result by SymTA/S is too pessimistic. In fact, the processor utilization is not sufficient for precise scheduling analysis because it is possible that the processor is not overloaded but a task misses its timing constraint. For example, few typing errors while modeling the system within SymTA/S modified the process-WCETs so that the 10ms-task missed its timing constraint but the maximum processor utilization determined by SymTA/S was about 98.1%, see report file *RPF03*. Even without typing errors, the 10ms-task's WCRT of 8943.11 $\mu$s is close to its timing constraint of 10 ms. This WCRT situation is illustrated by the Gantt-chart in figure 3.23.
The interpretation of the scheduling analysis result has to take several aspects into account. On the one hand, the combination of several measured maximum execution times, which is determined by SymTA/S, eventually does not occur in reality and leads to pessimistic results, see section 2.2.2. The used execution time measurement method causes overhead that affects the results. Section 2.6.2 also describes that in the example project, INCA



المنارة للاستشارات
www.manaraa.com

**Observed Paths**

| Path | # Events | Latency | Constraint | Buffer | Status |
|---|---|---|---|---|---|
| 1ms_Timer;o0 1ms;o | 1 | [34.3, 303.6] | 1000 | 1 | SUCCESS |
| 4ms_Timer;o0 4ms;o | 1 | [164.9, 767.02] | 4000 | 1 | SUCCESS |
| 10ms_Timer;o 010ms;o | 1 | [1914.45, 8943.11] | 10000 | 1 | SUCCESS |
| 20ms_Timer;o 020ms;o | 1 | [185.55, 7958.76] | 20000 | 1 | SUCCESS |
| 30ms_Timer;o 030ms;o | 1 | [733.9, 25897.53] | 30000 | 1 | SUCCESS |
| 50ms Timer: o050ms;o | 1 | [30. 25486.67] | 50000 | 1 | SUCCESS |
| 100ms_Timer; o0100ms;o | 1 | [97.25, 27576.57] | 100000 | 1 | SUCCESS |

Figure 3.22: SymTA/S scenario 1: observed paths using process execution times measured by INCA

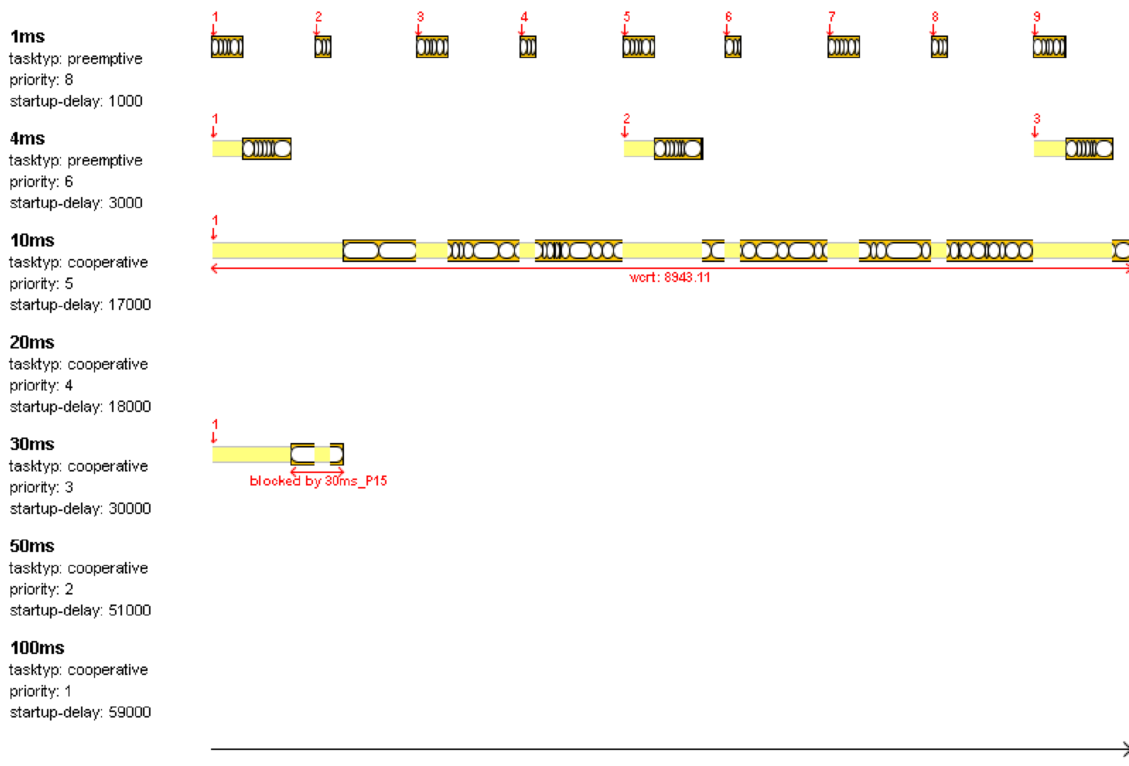



Figure 3.23: SymTA/S scenario 1: Gantt-chart of the 10ms-task's critical instant leading to its WCRT (the white bubbles are the task's processes)



المنارة للاستشارات
www.manaraa.com

measurements implicitly include ISRs into the measured execution times. Therefor, it is assumed that more ISRs are included through the given execution times that really occurred. Additionally, it is easily imaginable that another INCA measurement could provide (slightly) different results. Eventually, SymTA/S then determines a constraint violation of a task's WCRT, especially in a situation like here where a task just meets its constraint. On the other hand, it can not be guaranteed that the measured maximum execution times are safe, i.e. the WCET might not be measured but possibly occurs during normal operation. Eventually, the WCET of a certain process or task would be higher than the execution time measured by INCA which includes ISRs. This eventually might lead to an optimistic scheduling analysis.
In order to provide a very precise scheduling analysis result, the worst-case situation has to be measured that includes the combination of those execution times that leads to the WCRT of a certain task. However, the software is a black-box and so it is not practicable to identify and measure that worst-case situation due to the software's (and hardware's) complexity. Recording execution times while running a SoftCar RT-script (that causes high processor utilization) and using SymTA/S to determine a combination of execution times that potentially could lead to a scheduling problem is just an attempt to determine the worst-case situation which is practicable under the current circumstances. This approach does not qualify to be safe, i.e. it does not qualify to provide results that are equal to or more pessimistic than the worst-case situation for the reasons described above. Even if SymTA/S would determine a scheduling problem, this situation is not guaranteed to be the worst-case situation because the used maximum execution times measured by INCA are not guaranteed to be safe. In general and in particular for the example project, it is unknown if a potentially pessimistic combination of possibly unsafe WCETs results in higher WCRTs than in the (unknown) worst-case situation. However, if it would be assumed that the situation determined by SymTA/S is too pessimistic because it would be assumed that the determined execution time combination does not occur in reality, it has to be investigated if it can be guaranteed that this combination never occurs. Even if this would be possible despite the software's (and hardware's) complexity, the determination of the real worst-case situation with SymTA/S, which eventually leads to a scheduling problem, requires case-sensitive execution times that occur in this worst-case situation. At this point, black-box-tests like INCA measurements obviously are not suitable to gain that information. Such a case-sensitive execution time analysis requires knowledge about the software internals which are not known by a single (or few) ZF employee(s). Knowing the case-sensitive WCETs of the worst-case situation implies knowledge about the worst-case situation itself. So, SymTA/S is assumed to be not necessary then.
One possibility to shorten the WCRTs of several tasks is to move a process to a task with longer cycle time. SymTA/S allows testing this without time consuming software adaptation. In the current example, moving a process from the fastest task to a slower one reduces the WCRT of all tasks except the 20ms-task, see report file *RPF04* and compare with report file *RPF02*. Note that this process movement probably is not allowed, e.g. because the software within these tasks is supplied to ZF.



المنارة للاستشارات
www.manaraa.com

### Sensitivity Analysis

Theoretically, the sensitivity analysis can be used to determine a safety margin of a system that bases on unsafe (measured) WCETs. It might also be used in the case when a timing problem is identified and code modifications become necessary. SymTA/S could make the decision easier which process(es) to adapt by determining the WCET of each task respectively process that would be necessary to meet a specified processor utilization or a specified timing constraint, see section 3.3.6.
The effort that is necessary to perform a precise (case-sensitive) execution time analysis plus a well directed code modification eventually might be much higher than making just a code modification that bases on sensitivity analysis results that use imprecise maximum execution times. In every case, such a code modification requires a precise respectively a re-producible execution time measurement method. INCA is not suitable for this purpose, see section 2.6.2. It has to be decided if code modifications should be made although the scheduling and sensitivity analysis probably bases on unsafe WCETs. It becomes apparent again that black-box testing is not sufficient to gain very precise results respectively to make a right decision how to act in case of a scheduling problem indicated by SymTA/S.
A sensitivity analysis on the example system with the constraint of 97% processor utilization indicates no WCET tolerance for all processes after round 35 minutes because this processor utilization is already met and no further constraints are violated, see report file *RPF02*. It might be necessary to work with imprecise or unsafe WCETs, like in the current project, and to examine the necessary execution time reduction(s) in order to allow the 10ms-task meeting a WCRT of 8000 $\mu$s. The results of such a sensitivity analysis are included by report file *RPF05*. Here, the analysis takes round 2 hours on the test system. This increase of analysis time is caused by the different search intervals for the binary search (see section 3.3.6) due to the new constraint. As result, reducing the WCET of one of the fastest task's processes about 21.9 $\mu$s allows the 10ms-task to meet the timing constraint of 8000 $\mu$s, see figure 3.24. It is also possible to reduce the WCET of the 10ms-task's processes about 175.18 $\mu$s (see figure 3.24). This WCET tolerance can be distributed to the processes of the task. For example, a WCET reduction of 10ms_P1 about 80 $\mu$s, of 10ms_P3 about 35 $\mu$s and of 10ms_P6 about 60.18 $\mu$s allows meeting the WCRT constraint of the 10ms-task, see report file *RPF06*. In order to play safe, a subsequent scheduling analysis verifies that. Now, amongst others the 20ms-task has an increased WCET tolerance that still allows meeting the new WCRT constraint of the 10ms-task but also reaching a maximum processor utilization of 97%.
The sensitivity analysis result of the system under the assumption that a process is moved from the fastest to another task can be found in report file *RPF04*. SymTA/S takes nearly 11 hours and determines process WCET tolerances of at least 66.45% for a process of the second fastest task. This is the lowest WCET tolerance by far. Under that circumstances, the system would still met all timing constraints respectively the processor utilization is not above 97%.



المنارة للاستشارات
www.manaraa.com

WCET

| Task | Current WCET | Maximum WCET | Slack | Flexibility (%) |
|---|---|---|---|---|
| 1ms->1ms_P0 | 74.25 | 52.35 | -21.9 | -29.49% |
| 1ms->1ms_P1 | 39.5 | 17.6 | -21.9 | -55.44% |
| 1ms->2ms_P0 | 36.05 | 0.0 | NaN | |
| 1ms->1ms_P2 | 35.65 | 13.75 | -21.9 | -61.43% |
| 1ms->2ms_P1 | 74.15 | 30.35 | -43.8 | -59.07% |
| 1ms->2ms_P2 | 44.0 | 0.2 | -43.8 | -99.55% |
| 4ms->4ms_P0 | 111.05 | 23.46 | -87.59 | -78.87% |
| 10ms->10ms_P0 | 345.4 | 170.22 | -175.18 | -50.72% |
| 10ms->10ms_P1 | 368.95 | 193.77 | -175.18 | -47.48% |
| 10ms->10ms_P2 | 36.35 | 0.0 | NaN | |
| 10ms->10ms_P3 | 74.5 | 0.0 | NaN | |
| 10ms->10ms_P4 | 45.4 | 0.0 | NaN | |
| 10ms->10ms_P5 | 97.45 | 0.0 | NaN | |
| 10ms->10ms_P6 | 241.65 | 66.48 | -175.17 | -72.49% |

Figure 3.24: SymTA/S scenario 1: sensitivity analysis results with WCRT constraint of 8000 $\mu$s for the 10ms-task (from report file *RPF05*)

### Design Space Exploration

The design space exploration plug-in could give hints about possible system parameter optimizations. This might be necessary if a potential problem is identified. Even if there are no problems, the design space exploration is intended to help optimizing the system configuration, possibly in conjunction with code modifications, in order to shorten WCRTs. For demonstration, the exploration of the activation offsets of the current system is started with the initial task activation offsets. The synchronized sources on *OSEK-ECU* are specified as search space in order to optimize the activation offsets. Like described in section 3.4.2, the smallest modification unit is set to 100 because SymTA/S does not provide a higher one. For the current example, a smallest unit of 1000 is required because the offsets are only adjustable in units of 1 ms. For a first optimization run, the 10ms-task's timing constraint of 10000 $\mu$s and the minimization of parameter changes are specified as optimization objectives. All further settings are default (seed: 1201090819042). SymTA/S determines 2 pareto-optimal configurations (duration: 1:45 hours). One of these configurations is the initial configuration. The new configuration allows the 10ms-task meeting a WCRT of 8175.17 $\mu$s, see enclosed report file *RPF07*. The activation offsets determined by SymTA/S are rounded (mathematically) in order to represent a real system configuration for subsequent scheduling analysis. Then, the 10ms-task's WCRT is 8943.11 $\mu$s, so that there is no improvement at all (see report file *RPF08*). Obviously, rounding the activation offsets to multiples



المنارة للاستشارات
www.manaraa.com

of 1000 changes the WCRT optimizations in some cases, see table 3.1. Thus, an appropriate time basis should be used in order to be able to use the system configurations that are determined by SymTA/S as seen so that manually checks could be avoided.

| opt. objectives<br>10ms-task's constraint [$\mu$s]<br>report file | 10ms-task<br>10000<br>*RPF07(08)* | 10ms-task<br>8500<br>*RPF09(10)* | all tasks<br>8500<br>*RPF11(12)* |
|---|---|---|---|
| fastest task [$\mu$s] | 303.6 (303.6) | 303.6 (303.6) | 303.6 (303.6) |
| fast task [$\mu$s] | 667.02 (767.02) | 612.82 (612.82) | 612.82 (612.82) |
| 10ms-task [$\mu$s] | 8175.17 (8943.11) | 8175.17 (8479.69) | 8175.17 (8788.91) |
| 20ms-task [$\mu$s] | 6072.5 (6572.5) | 5967.98 (4958.78) | 5867.98 (6572.5) |
| 30ms-task [$\mu$s] | 23860.95 (24511.27) | 21006.75 (20912.29) | 21906.75 (22511.27) |
| 50ms-task [$\mu$s] | 23450.09 (23950.09) | 29150.09 (28950.09) | 21650.09 (21950.09) |
| 100ms-task [$\mu$s] | 53792.42 (54596.94) | 56692.42 (36598.72) | 42192.42 (39442.74) |
| changed offsets | 2 (1) | 5 (5) | 5 (4) |
| $\sum$ WCRT [$\mu$s] | 1018136.4 (1068091.36) | 1026861.6 (958433.2) | 934937.54 (917792.14) |

Table 3.1: SymTA/S scenario 1: WCRTs resulting from design space exploration, values in brackets results from rounded activation offsets (compare with initial system configuration (figure 3.22), $\sum$ WCRT: 1030960.23 $\mu$s)

In order to get a better result, a second optimization is performed that starts with the initial configuration too. However, the 10ms-task's timing constraint is set to 8500 $\mu$s because the exponential miss penalty for a missed constraint is activated by default. So, the 10ms-task's WCRT becomes a high optimization priority because this constraint is not fulfilled in the initial configuration. SymTA/S determines 1 pareto-optimal configuration (duration: 1:55 hours). Unmodified, it allows the 10ms-task meeting a WCRT of 8175.17 $\mu$s, see report file *RPF09*. After rounding the activations offsets to a representative configuration, the 10ms-task has a WCRT of 8479.69 $\mu$s. Here, rounding the activation offsets lengthens only the 10ms-taks's WCRT. All other resultant WCRTs are equal or better than that determined by SymTA/S. However, the 50ms- and 100ms-task's WCRT is worsen in comparison to the initial system configuration, see report file *RPF10* and compare with report file *RPF02*.
In order to investigate this further, a third optimization is performed. All task timing constraints (10ms-task: 8500 $\mu$s) and the minimization of parameter changes are specified



المنارة للاستشارات
www.manaraa.com

as optimization objectives. SymTA/S determines 9 pareto-optimal configurations. Three of them provide a WCRT of 8325.49 $\mu$s for the 10ms-task after rounding the activation offsets to a representative configuration. One of those three configurations provides the lowest weighted sum of all task WCRTs within the hyperperiod of 300 ms (after rounding). It is determined by the sum over all task WCRTs that are multiplied with the number of their activations within the hyperperiod. Note that these weighted WCRTs do not qualify to represent the overall WCRT of all tasks within the hyperperiod. One the one hand, a task has its WCRT not every time it is activated. On the other hand, the fastest task is activated more often than the 100ms-task and so a middle fastest task's WCRT might affect the overall WCRT more than a low 100ms-task's WCRT. It is one possibility to rate the system configurations. These weighted WCRTs of all investigated configurations are listed in table 3.1. Scheduling analysis results using this special configuration from the third optimization can be found in report files *RPF11* (not rounded) and *RPF12* (rounded). In comparison with the initial system configuration (see figure 3.22), only the 50ms-task's WCRT is lengthened whereas all other tasks have better or equal WCRTs.
Surprisingly, an optimization which considers all task constraints provides the best 10ms-task's WCRT at all after rounding. However, the optimizations with only the 10ms-task's WCRT as optimization objective provided the same 10ms-task WCRTs like this configuration before rounding. So, rounding obviously causes precision loss and the need for using an appropriate time base becomes apparent. Otherwise, manually checks of the optimization results are necessary in order to determine the best system configuration. The comparison of the weighted sums of task WCRTs indicates that the more constraints have to be optimized the more the average WCRTs are reduced. Further, setting the timing constraints appropriate allows directing the optimization, e.g. on a special task WCRT. Subsequent sensitivity analyses of all configurations do not determine higher WCET tolerances than before because the specified constraint of 97% processor utilization is still met. A higher sensitivity analysis constraint would increase sensitivity analysis time and might be unsuitable because a maximum allowed processor utilization of for example 99% might be within the frequency fluctuation range of a real processor.

### 3.4.4 Conclusion of Scenario 1

The above described analyses clarify that the scheduling analysis that bases on the maximum execution times derived from hardware traces or measured by INCA is not sufficient for system verification. It is late in development i.e. in the integration phase of the V-model. INCA as well as the debugger method have weak points. Obviously, the scheduling analysis that bases on the given hardware-trace does not catch the worst-case situation. The hardware-trace is complex to analyze precisely and complicates the SymTA/S model because the ISRs have to be considered additionally. It is not possible to give a statement if the scheduling analysis described in section 3.4.3 represents a safe respectively pessimistic or an optimistic result. A scheduling analysis that bases on another INCA measurement of the same SoftCar RT-script possibly could lead to different results, see



المنارة للاستشارات
www.manaraa.com

section 2.6.2. The difficulty to determine the worst-case situation is rooted in the black-box character of the software. It is assumed that shorter INCA-measurement respectively longer hardware-trace recording times do not improve the situation because it is necessary to catch the worst-case situation respectively the worst-case combination of process execution times for very precise scheduling analysis.
In case of a slightly missed scheduling analysis, measuring the cycle time of the appropriate task might show that the task does not miss task activation. This possibly satisfies a costumer although this does not guarantee the absence of a possible scheduling problem. It is assumed that problems which occur due to too long execution times would be detected earlier during conventional tests without the need for a special scheduling analyzer. This also holds for the current example when the dynamic load like ISRs would not be considered. SymTA/S might be used to examine the affect of dynamic load scenarios on the system. However, if ISRs are already included, it is hard to determine several scenarios.
If the timing behavior of the software must be guaranteed, one possibility is to determine the WCET of manageable code pieces, e.g. single functions, processes and ISRs, and then verify the schedulability by SymTA/S while assuming several ISR occurrence scenarios. However, using only (safe) WCET assumably requires the use of hardware that is more powerful than necessary because the scheduling analysis result potentially is too pessimistic. In return for these pessimistic assumptions, one gets a safe timing behavior of the system given an accurate WCET and scheduling analysis. A possibility to make (in particular dynamic) WCET analysis on black-box software less difficult is to change the programming style. The execution time of the code should be as constant as possible, like described in section 2.6.5. This reduces its execution time-related complexity. This approach is only advantageous if the hardware-sided interdependencies on the execution time are negligible in comparison to the software-sided interdependencies. This new programming style might be applied stepwise in development process of ZF.
The sensitivity analysis might be helpful in case of safely identified problems. It could make the determination of system parameters to be modified less difficult. The design space exploration also might be required in case of safely identified serious problems. However, changing parameters like activation offsets or even priorities is improbable in a late development phase.
Modeling the system within SymTA/S, performing and evaluating the analysis are straightforward. The input of numerous process execution times is time-consuming as each task and source has to be added to the SymTA/S GUI. A script-supported input should be preferred. Knowledge about the system configuration concerning activation offsets and priorities is needed. Analysis can be performed by just one click. It takes several seconds on the test system. The results are easy to interpret because no complicated assumptions have to be made by the analysis tool.



المنارة للاستشارات
www.manaraa.com

## 3.5 Scenario 2: Extension of an Existing System

### 3.5.1 Preface

The second scenario to be examined concerns the analysis of an existing system which is extended by a further functionality. Figure 3.25 illustrates the extended system. It is

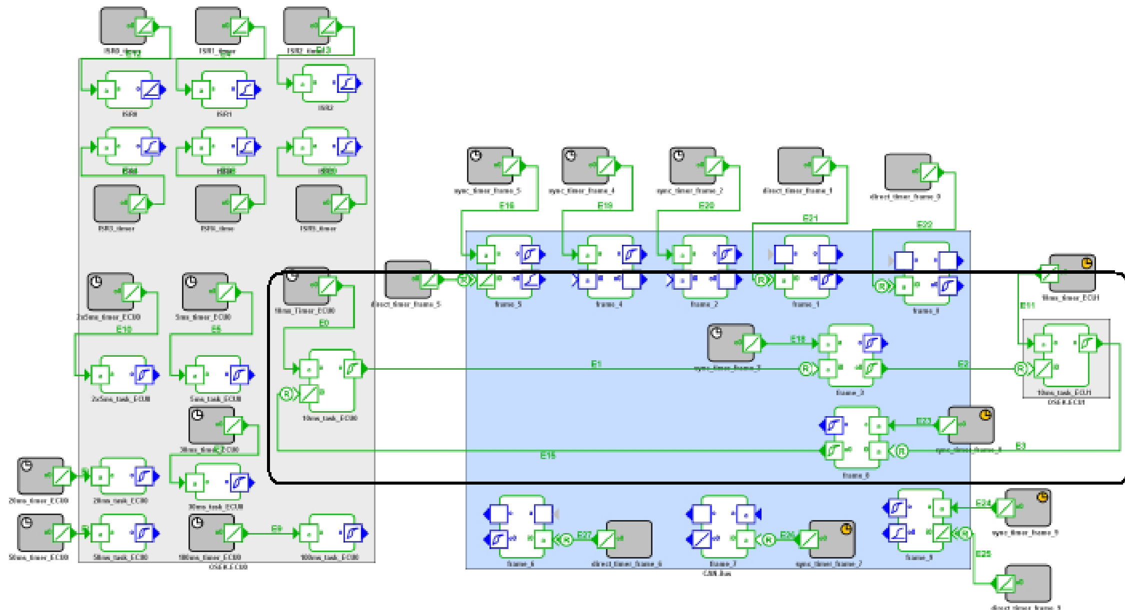


Figure 3.25: SymTA/S scenario 2: system model, the surrounded area represents the new functionality to be investigated

intended to investigate the timing behavior of the communication with another ECU. The involved system elements are surrounded by the black border within figure 3.25. Therefor, data is written by the 10ms-task on *OSEK-ECU0* (left). That data is sent via a CAN-bus (middle). Then, the data is read by the 10ms-task on *OSEK-ECU1* (right). Finally, its output is send back to the 10ms-task on *OSEK-ECU0* via the CAN-bus. It should be examined which parameters affect the cycle WCRT respectively how many activations the 10ms-task on *OSEK-ECU0* has to wait until the sent data reaches it, e.g. in scope of a handshake protocol.

### 3.5.2 System Description

*OSEK-ECU1* has a NEC V850 processor which runs an OSEK OS implementation. According to the corresponding OIL-file, the system provides several ISRs and tasks. The tasks are set to be non-preemptive and can be activated only once per period. One of them is activated periodically every 10 ms. However, no further activation assumptions can be



للاستشارات المنارة
www.manaraa.com

made. Unfortunately, the existing software currently does not implement the functionality described above. So, the WCET of the 10ms-task on *OSEK-ECU1* is estimated to be between 8 and 9.5 ms. That should simulate the possibly jittering WCRT of this task including all blocking times and preemptions by interrupts and higher priority tasks. More detailed information is not available because the ECU is supplied to ZF. No execution time analysis is performed on this ECU.
*OSEK-ECU0* runs the task set like described in section 3.2.2. Additionally, six interrupts are considered, see figure 3.25. The used WCETs of the tasks and of these six ISRs on *OSEK-ECU0* are measured by INCA while running a SoftCar RT-script that causes high processor utilization. Because not all functionalities are implemented, the measured execution times are assumed to increase in later software versions. So, the current maximum execution times are an assumption. In contrast to section 3.4.3, here SymTA/S provides measuring only a few processes at the same time. Their start addresses have to be assigned manually to predefined measurement variables. Although precise scheduling analysis requires process execution times, only the task execution times are measured because they are assumed to be sufficient for investigation of the described timing behavior which is described above. These task execution times do not contain the execution times of the six ISRs that can be measured too. The sum of these ISR execution times and number of activations are determined over an interval of 100 ms. So, the scheduling analysis just uses average ISR execution times and activation frequencies. The modeled system is not as precise as possible but it can be created quickly. The accuracy is sufficient because pessimistic assumptions are made later and (timing-) information about *OSEK-ECU1* are rare. Precise but time-consuming measurements would have to be performed every time a new functionality is implemented.
The CAN-bus is modeled by importing a DBC-file via the SymTA/S interface. The CAN-bus speed is assumed to be 500 kBits/s. It transmits 10 frames which have transmission modes periodic, direct or mixed. The CAN-frames of interest are *frame_3* and *frame_8*. Both are sent periodically every 10 ms and have a payload of 8 bytes. They are carrying the data for the handshake protocol. Therefor, both frames are connected to the involved tasks by event streams. So, it is a signal-path end-to-end analysis and not an event-triggered one.
The activation sources of the periodic and mixed CAN-frames are synchronized with the task activations on the corresponding ECU. Also the task activations on each ECU are synchronized. This is realized by two "clocks" within the system. One for *OSEK-ECU0* and one for *OSEK-ECU1*. Unfortunately, information about the activation offsets is not (yet) available. So, they are assumed to be zero, i.e. all task and CAN-frames are activated simultaneously. Although this approach is pessimistic, it gives a safety margin to the WCRT of the involved tasks. It is assumed that both tasks have a timing constraint of 10 ms. So, an activation offset of zero for the CAN-frames also works if the tasks have a high WCRT but do not violate their constraints. More sophisticated activation offsets probably shorten the task WRCTs and minimize the data's waiting time in the CAN-frame buffer. For a first analysis, this might be sufficient because SymTA/S allows trying several



المنارة للاستشارات
www.manaraa.com

configurations very quickly.

### 3.5.3 Scheduling Analysis

Scheduling analysis determines a WCRT of 41.388 ms until the data can be read again by the 10ms-task on *OSEK-ECU0* while using the *First Through* semantic. The cycle WCRT Gantt-chart is illustrated by figure 3.26 respectively the enclosed cycle Gantt-chart "[RPF13] scen2-zero-firstthrough-normal_WCET". SymTA/S combines the local WCRTs of the involved tasks and CAN-frames like described in section 3.3.5. Each local WCRT and the corresponding Gantt-chart can be taken from the enclosed report file *RPF13*.
The WCRT of the path is determined like described in the following. SymTA/S determines a WCRT of 6.556 ms for the 10ms-task on *OSEK-ECU0* during local scheduling analysis. The data to be sent waits 3.444 ms in the buffer of *frame_3*. The frame is activated to the next full 10 ms because the activation offsets are zero. The transmission by *frame_3* adds its WCRT of 0.543 ms. Now, the analysis leaves a synchronized group and SymTA/S assumes a maximum delay of 10 ms on event-stream *E2*. *OSEK-ECU1* and *frame_8* form the next synchronized group. The 10ms-task on *OSEK-ECU1* is assumed to have a WCRT of 9.5 ms. Due to the activation offsets, the data waits 0.5 ms until *frame_8* is activated which adds a WCRT of 0.804 ms. After transmission by *frame_8*, SymTA/S adds a delay of 10 ms on event-stream *E15* because a synchronized group is left. Hence, SymTA/S determines a cycle-WCRT of 41.388 ms. In fact, the data can be read 40 ms after the corresponding activation of the 10ms-task on *OSEK-ECU0*. The reason for this pessimistic delay assumption on event-stream *E15* can be traced back to the fact that the history about the synchronization with the starting point is not recorded by SymTA/S. Figure 3.27 summarizes the cycle WCRT. The first WCRT of 31.338 ms is the WCRT of the path that ends after transmission of *frame_8*. It does not take the last delay on event-stream *E15* into account. This clarifies that the data has to wait just 8.612 ms instead of 10 ms on event-stream *E15*.
Because the ECUs are not synchronized to each other, data could be lost during communication. Assume the case where the 10ms-task on *OSEK-ECU1* reads the data immediately after *frame_3*, which has its BCRT, has written it to the buffer. In a hypothetical case, the frame-transmission jitters the next time i.e. the buffer is written after the 10ms-task on *OSEK-ECU1* read it. In the next cycle, the CAN-frame does not jitter and the data is overwritten before it is read. So, there are two instances of the first data in the system whereas the second instance is missed. Like described in section 3.3.5, it is interesting how many instances of the same data could traverse the path. This has to be considered when implementing a handshake protocol. SymTA/S helps detecting this under-sampling effect by using its *Maximum-Age* semantic for signal-path analysis. It indicates a greater delay on the corresponding event-streams. The results from path analysis using this semantic are included by report file *RPF14* and the Gantt-chart "[RPF14] scen2-zero-maxage-normal_WCET" within the enclosed archive. However, it does not help to determine the number of data instances within the system. Such under-sampling effects might occur at



المنارة للاستشارات
www.manaraa.com

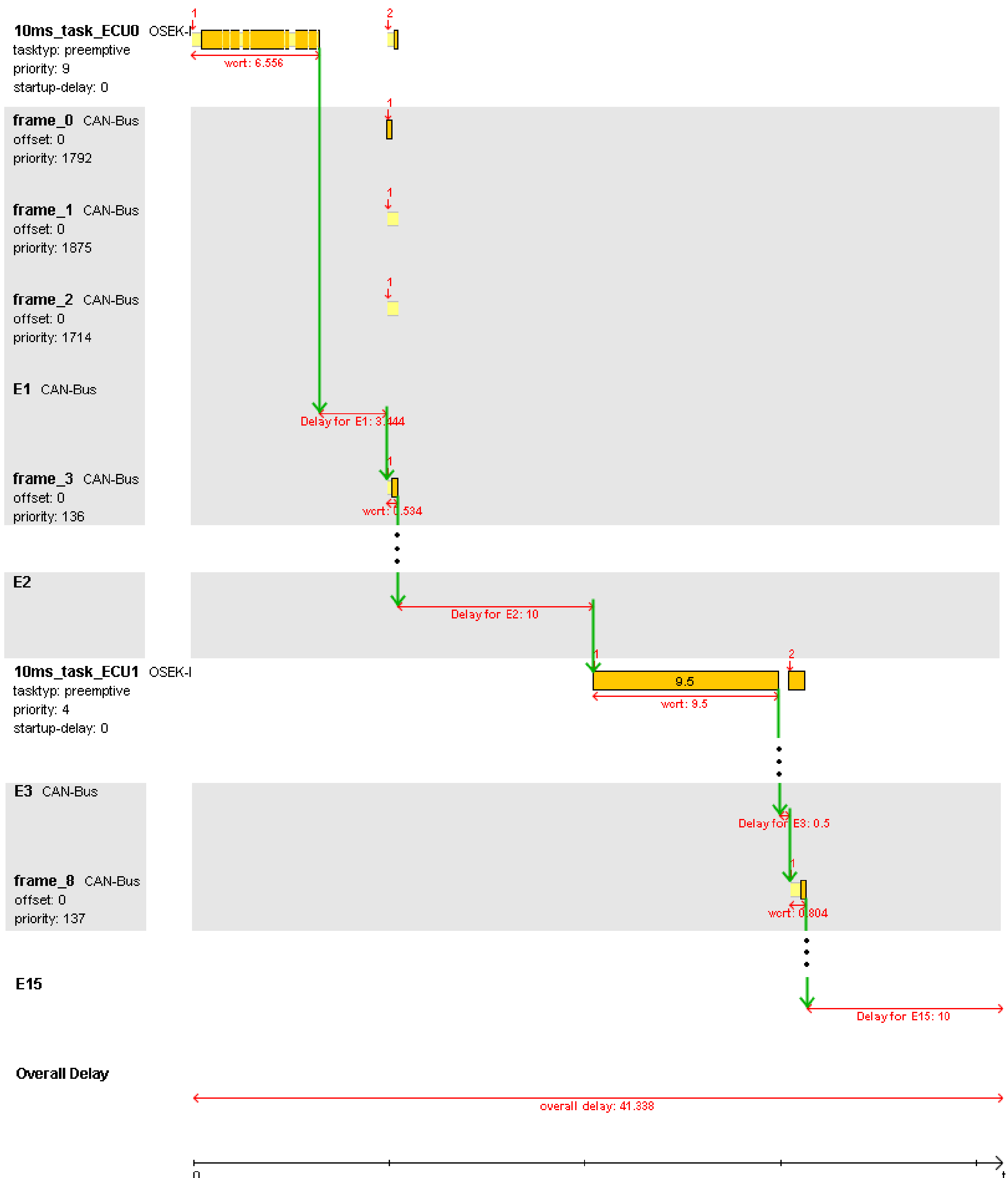



Figure 3.26: SymTA/S scenario 2: cycle Gantt-chart extract which does not display all blocking CAN-frames due to readability reasons



www.manaraa.com

**Observed Paths**

| Path | # Events | Latency | Constraint | Buffer | Status |
|---|---|---|---|---|---|
| 10ms_Timer_ECU0;o0frame_8;o0 | 1 | [0.432, 31.338] | - | 8 | SUCCESS |
| 10ms_Timer_ECU0;o010ms_task_ECU0;i0 | 1 | [0.432, 41.338] | - | 9 | SUCCESS |

Figure 3.27: SymTA/S scenario 2: cycle WCRT using *First Through* semantic (the first WCRT does not take the delay determined by SymTA/S between *frame_8* and the 10ms-task on *OSEK-ECU0* into account)

each point where jitter occurs. It is supposed that SymTA/S would not determine that the under-sampling effect in the example does not occur on *frame_3* and *frame_8* at the same time under the current system configuration. Again, the reason it that it does not record synchronization history between synchronized groups. So, this has to be checked manually.

### 3.5.4 Sensitivity Analysis

In projects like this where precise parameters are unknown, the sensitivity analysis of SymTA/S might be useful. Unfortunately, for this example it is not useful because the WCET of the 10ms-tasks on *OSEK-ECU0* and *OSEK-ECU1* does not directly affect the cycle WCRT in the current configuration. Sensitivity analysis does not determine the sensitivity of activation offsets. So, its range of use is limited in this case. An analysis could only determine the minimum period for the ISRs until the processor is overloaded or a timing constraint is missed.
In order to clarify that system optimization could lead to less flexibility, a sensitivity analysis of the system is performed. The analysis settings are a maximum processor utilization of 97% and timing constraints for all tasks of *OSEK-ECU0* that are equal to their period. First, the task WCET tolerances of *OSEK-ECU0* are analyzed with the initial system configuration, i.e. all activation offsets are zero. SymTA/S determines the same WCET tolerances no matter if using the *First Through* or *Maximum Age* semantic, see figure 3.28. The results can be found in the enclosed report files *RPF13* (*First Through* semantic) and *RPF14* (*Maximum Age* semantic). Here, the specified maximum processor utilization of 97% on *OSEK-ECU0* is the crucial factor for determining the WCET tolerance. Assigning the maximum WCET to the 10ms-task (7.92 ms) that is determined by SymTA/S allows the 10ms-task still meeting its WCRT but the processor utilization is about 97%, see figure 3.29 or the enclosed report file *RPF15*. However, setting the 10ms-task's WCET to 8.2 ms leads to a maximum processor utilization about 99.8% while the 10ms-task still meets



المنارة للاستشارات
www.manaraa.com

WCET

| Task | Current WCET | Maximum WCET | Slack | Flexibility (%) |
|---|---|---|---|---|
| 10ms_task_ECU0 | 5.33 | 7.92 | 2.59 | 48.59% |
| 2x5ms_task_ECU | 0.11 | 0.75 | 0.64 | 581.82% |
| 5ms_task_ECU0 | 0.2 | 1.49 | 1.29 | 645% |
| 20ms_task_ECU0 | 0.23 | 5.41 | 5.18 | 2252.17% |
| 30ms_task_ECU0 | 0.2 | 7.97 | 7.77 | 3885% |
| 50ms_task_ECU0 | 0.12 | 13.08 | 12.96 | 10800% |
| 100ms_task_ECU 0 | 0.4 | 26.33 | 25.93 | 6482.5% |

Figure 3.28: SymTA/S scenario 2: sensitivity analysis results for *First Through* and *Maximum Age* semantic

OSEK-ECU0

| Speed Factor | 1.0 |
|---|---|
| Context Switch Time | 0 ÷ 1.0 = 0 |
| Scheduling | Generic OSEK |
| Requirements | any |
| Utilization | 97.0% |
| Scheduling Overhead | 0.0% |
| Kernel Priority | 16 |
| Activation Buffer | 1 |
| Up-To-Date | yes |
| Resource Status | SUCCESS |
| Selected Analysis | Full Offset Analysis |

Figure 3.29: SymTA/S scenario 2: processor utilization after assigning the maximum WCET to the 10ms-task on *OSEK-ECU0* that is determined by sensitivity analysis using the *First Through* semantic



المنارة للاستشارات
www.manaraa.com

its WCRT constraint, see enclosed report file *RPF16*. The same holds for the analysis using the *Maximum Age* semantic, see report files *RPF17* (10ms-task WCET: 7.92 ms) and *RPF18* (10ms-task WCET: 8.2 ms). The results of sensitivity analysis using an optimized system configuration are described in the next section.
In the case where a further process should be added to the system, the sensitivity analysis could determine the execution time tolerance within the designated task. Like described in section 3.4, it determines the highest possible execution time of the current code which still allows meeting specified timing-constraints or processor load. This tolerance can be compared with the new code's WCET. If it is less or equal to it, it can be assigned to the system and the system constraints should be verified by a subsequent scheduling analysis. Like described in the antecedent section, the determination of the WCET has to take several issues into account. If the WCET is above the WCET tolerance, the sensitivity analysis helps to determine which process(es) to adapt in order to allow including the new code.

### 3.5.5 Design Space Exploration

The design space exploration plug-in could be used to find a less pessimistic system configuration for the example system. The activation offsets are not known at time of this work. So, the optimization functionality quickly provides one possible, more realistic configuration. Optimization objective is the WCRT of the end-to-end path from the 10ms-task on *OSEK-ECU0* to *OSEK-ECU1* and return. In order to choose this path as optimization objective, a constraint of 40 ms is specified within SymTA/S.
The smallest unit for offset changes is set to five although that is below the exact offset resolution. All further settings are default (seed: 1201090819042). The optimization starts with the initial configuration (all offsets zero). As search space, the synchronized sources on *OSEK-ECU0* are specified. *OSEK-ECU1* is not specified as search space because event stream *E3* between the 10ms-task on *OSEK-ECU1* and *frame_8* contributes only 0.5 to the cycle WCRT and information are very rare for that ECU. Unfortunately, SymTA/S does not find a configuration that reduces the cycle WCRT to 40 ms or better. That is not hard to see because the offset between the 10ms-task and *frame_3* had to be set to five. Then, the WCRT of the 10ms-task has to be reduced to five or better. Obviously, this is not possible. So, a further run with a smallest offset change unit of one, which is below the real offset resolution, is performed. After approximately one hour, the design space exploration plug-in determines one pareto-optimal configuration, see table 3.2. It reduces the cycle WCRT to 38.338 ms, i.e. the data can be read 30 ms after the corresponding activation of the 10ms-task on *OSEK-ECU0*. Figure 3.30 respectively the enclosed Gantt-chart "[RPF19] scen2-opt-firstthrough-normal_WCET" illustrate the optimized cycle WCRT.
The optimized system configuration affects the sensitivity analysis and leads to a reduced task WCET flexibility. When using the optimized system configuration and *First Through* semantic (report file *RPF19*), SymTA/S determines a lower WCET tolerance for the 10ms-task on *OSEK-ECU0* (0.74 ms) than when using the *Maximum Age* semantic (2.59 ms,



المنارة للاستشارات
www.manaraa.com

report file *RPF20*). It is assumed that in the first case, the transmission of *frame_3* is the crucial factor. The WCRT of the 10ms-task has to meet the frame transmission.

| | old activation offset [ms] | new activation offset [ms] | old WCRT [ms] | new WCRT [ms] |
|---|---|---|---|---|
| fastest task | 0 | 1 | 0.2954 | 0.2954 |
| fast task | 0 | 1 | 0.4954 | 0.4954 |
| 10ms-task | 0 | 4 | 6.556 | 6.236 |
| 20ms-task | 0 | 11 | 6.786 | 0.7348 |
| 30ms-task | 0 | 18 | 6.9954 | 2.436 |
| 50ms-task | 0 | 29 | 7.1364 | 1.566 |
| 100ms-task | 0 | 84 | 7.8008 | 6.846 |
| CAN-frame_3 | 0 | 1 | 0.534 | 0.534 |
| cycle | - | - | 41.338 | 38.338 |

Table 3.2: SymTA/S scenario 2: activation offset optimization results for *OSEK-ECU0*

The enclosed report file *RPF21* and the Gantt-chart extract in figure 3.31 (from enclosed Gantt-chart "[RPF21] scen2-opt-firstthrough-max_WCET") illustrates the situation where the possible maximum WCET of the 10ms-task (6.07 ms) is assumed. A higher WCET (e.g. 6.3 ms) would cause missing the intended frame transmission, see report file *RPF22* and the Gantt-chart extract in figure 3.32 from enclosed Gantt-chart "[RPF22] scen2-opt-firstthrough-over_WCET".
If *Maximum Age* semantic is used, the crucial factor is the processor utilization constraint of 97% like for the not-optimized system configuration. The enclosed cycle Gantt-chart "[RPF23] scen2-opt-maxage-max_WCET" illustrates the situation where the maximum WCET of the 10ms-task (7.92 ms) is assumed. See also report file *RPF23*. The data to be sent misses its intended transmission frame and takes the next frame which originally is intended for the next 10ms-task activation. If the WCET would be higher (e.g. 8.0 ms), the 10ms-task would still meet the task WCRT constraint but the processor utilization is above 97%, see enclosed report file *RPF24* and the enclosed Gantt-chart "[RPF24] scen2-opt-maxage-max_WCET".
The optimization also can be used in the case where a further process should be added to the system. It could determine necessary system changes in order to include the new code. So, it helps to trade off the effort to include new code which would be necessary if significant system parameter changes are required.

### 3.5.6 Conclusion of Scenario 2

The use of SymTA/S within a certain development phase in scope of system extension and functionality-shifting highly depends on the WCET analysis of the new or shifted code.



المنارة للاستشارات
www.manaraa.com

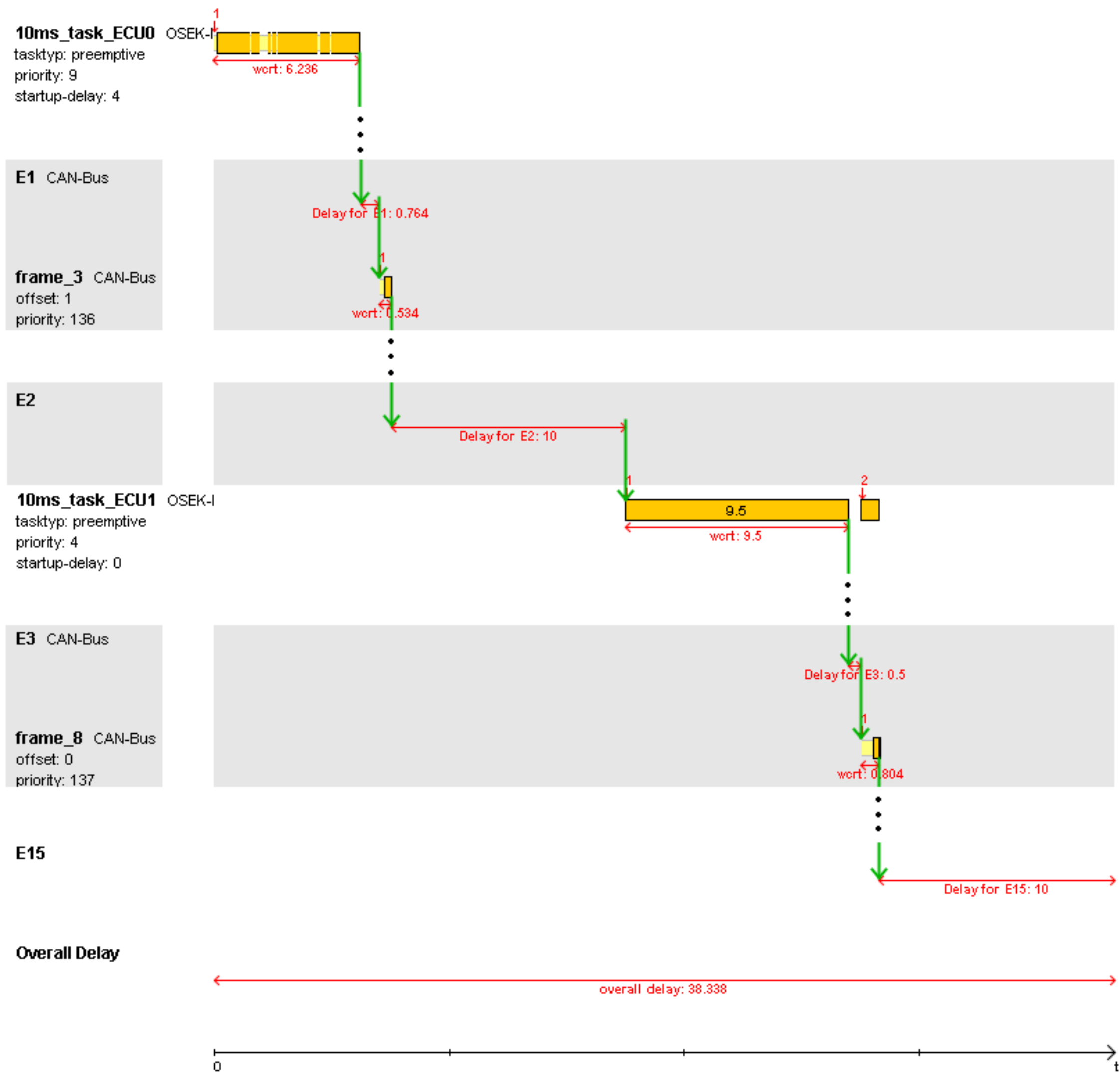



Figure 3.30: SymTA/S scenario 2: end-to-end path Gantt-chart extract from scheduling analysis with optimized configuration *First Through* semantic (blocking CAN-frames are not displayed due to readability reasons)



المنارة للاستشارات
www.manaraa.com

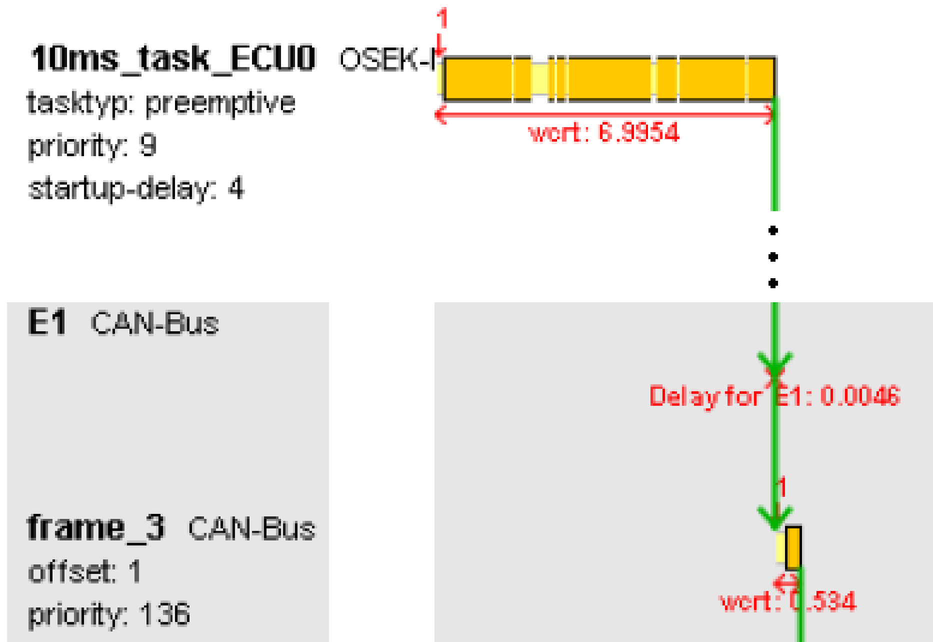



Figure 3.31: SymTA/S scenario 2: Gantt-chart extract where the maximum WCET of the 10ms-task on *OSEK-ECU0* is assigned to it while using the optimized system configuration and *First Through* semantic (blocking CAN-frames are not displayed due to readability reasons)

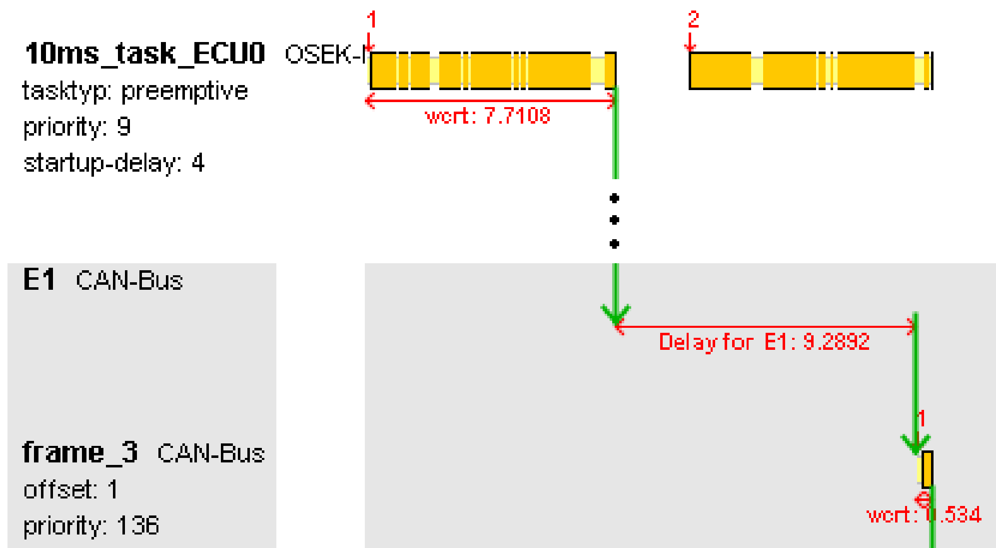



Figure 3.32: SymTA/S scenario 2: Gantt-chart extract where a WCET of the 10ms-task on *OSEK-ECU0* is assigned to it that is greater than the maximum possible while using the optimized system configuration and *First Through semantic* (blocking CAN-frames are not displayed due to readability reasons)



www.manaraa.com

If it would be possible to tightly examine the new code's WCET by using similar code respectively code with a similar functionality on another ECU, this can be made even before implementation phase. CAN-scheduling analysis just needs the bus-configuration, so that a first scheduling analysis could be performed before implementation. This first analysis might be pessimistic because the activation offsets of periodic task eventually are not known and have to be assumed to be equal for each frame. If the ECU-WCETs can not be examined, precise scheduling analysis with SymTA/S can be performed only late in the development process, i.e. in the function respectively module test phase or even first in the integration phase. That also depends on the general development phase of the system. If not all functionalities are implemented yet, the system has more capacity. Possible problems might just turn out late in development phase when the remaining functionalities are implemented.
In an existing system, the sensitivity analysis allows determining a possible WCET tolerance. It has to be checked if it is sufficient for new code. The existing system might be analyzed precisely before in order to avoid too pessimistic or too optimistic results. The design space exploration only might give proposals to solve serious problems. SymTA/S allows checking different system configurations quickly. So, the example illustrated by figure 3.25 might be analyzed under the assumption that a FlexRay- instead of a CAN-bus is used. If precise WCETs (and WCRTs) would be available, SymTA/S could be used as optimization tool, e.g. for a ZF-private FlexRay-bus.
The analysis of the SymTA/S scheduling analysis results has to be made carefully. Small changes in the system might lead to increased jitter that affects sampling effects in the system. As SymTA/S does not record the synchronization history, it is necessary to check if indicated sampling effects within the system could occur simultaneously. An improved BCRT analysis and an automatic possibility to determine how many data instances could traverse the path through a system are desirable.
It is not possible to simply add an existing SymTA/S system to another one. E.g. the CAN-bus is imported and a previously analyzed ECU should be added in order to generate a complex system model. One can create the system model via GUI again which is very time consuming. Another possibility is to combine the XML-files of both systems via copy-and-paste. This is less time-consuming but error-prone.

## 3.6 Scenario 3: Timing Budgeting

### 3.6.1 Timing Budgeting

The third scenario has a quite theoretical character. It should be examined if the specification of a timing budget in an early development phase, e.g. in system design phase of the V-model, is useful and how the timing budget could be met. The antecedent sections turn out that for verification respectively virtual extension of an existing system very precise input values or, if those are not ascertainable, pessimistic input values have to be used



المنارة للاستشارات
www.manaraa.com

in order to guarantee a certain timing behavior of a system. Deriving WCET estimations for certain functions, processes, tasks or functional groups of the software is quite difficult. Simple cross-estimations with different hardware architectures are not suitable, like described in section 2.6.5. An exception might be the following fictitious example. The system described in section 3.4 has to be extended with further functionality. The current ECU with TriCore 1766 is assumed to be not sufficient. So, the system's timing behavior should be determined if the software would run on a TriCore 1796. Both processors have a comparable architecture, see section 2.6.6. The (worst-case) execution times of the existing code derived on the TriCore 1766 might be multiplied with the ratio of both processor frequencies, whereas the TriCore 1766 has a frequency of 80 MHz and the TriCore 1796 of 150 MHz. However, it has to be investigated on the concrete hardware if the memory accesses of the ECUs with both processors have the same speed-up like the processor frequencies. If not, the overall speed-up could be limited by the memory accesses so that the conversion factor has to be reduced. Afterwards, sensitivity analysis could determine the execution time tolerance of the system that is available for the new functionality's code. This available execution time, less an optionally specified reserve, has to be distributed to the code that has to be implemented. This reserve might be kept ready in order to satisfy an increased timing budget demand later or to fulfil the resource requirements of further product generations that use the same ECU but provide more functionality. Sensitivity analysis only determines the maximum WCET of a process respectively task. So, timing budgeting is assumed to be an iterative process. After specifying the timing budget for a process or a part of the new functionality within a certain task, the system has to be analyzed again in order to determine the remaining available execution time within other processes/tasks. Eventually, a former specified budget or the reserve has to be reduced in order to accommodate the system. So, these budget specifications might be very time-consuming. Obviously, timing budgeting has to be made by an expert because they should be close to the WCETs of the (new) code. Otherwise, it might be the case that functions/processes do not meet their timing budget because the execution time might increase in later development phase due to necessary code modifications. These additional requirements might be satisfied by the kept ready reserve. Other software parts might not exhaust their budget. This not exhausted timing budget possibly should not be used to satisfy another requirement in order to avoid that each software developer writes code that exhausts its timing budget in every case. Specifying timing budgets should motivate the software developer to take the execution time of his code more into account.

This timing budgeting obviously becomes more difficult if no code is available at all. If a certain code basis is re-used for new projects, execution time estimations respectively a timing budget could be derived from former execution time investigations if similar hardware is used, like in the fictitious example which is described above. The situation even gets worse if there are no comparable projects. So, WCETs have to be estimated "well defined" whereas the budgeting expert is assumed to not know the code. Scheduling analysis is most precise when using process execution times if cooperative scheduling is used. However, it might be not manageable to specify the WCET of single functions



المنارة للاستشارات
www.manaraa.com

or processes. The more the execution time is distributed to small units, e.g. processes, the more pessimistic the timing budget might have to be specified because the budget-verification might be less case-sensitive than when specifying a timing budget for a whole functional group of the software. This especially might hold for complete new systems where no experiences from similar projects are available. However, case-sensitive WCET analysis is very complex or might not be practicable at all, like described above.
Remember the fictitious example which is described above. WCETs for the basic software's tasks or processes could be derived from the existing system with the TriCore 1766 processor. Then, SymTA/S allows analyzing several system configurations, i.e. which process is assigned to which task (period) or which task periods should be used at all. Again, the sensitivity analysis and design space exploration could help finding a configuration that provides low WCRTs and is flexible at the same time. However, this assumes more or less precise maximum execution time estimations. If there is no information available at all, it is also possible to make assumptions about several system configurations. If using a certain well defined system configuration, the newly implemented code has to be more or less suitable to this configuration (under consideration of a kept ready reserve) in order to avoid a system (configuration) re-design if the new code does not fit.

### 3.6.2 Control of the Timing Budget

The verification of meeting a given timing budget is not trivial. Like described in the section 3.6.1, specifying timing budgets for small units is assumed to be more pessimistic than specifying timing budgets for e.g. whole tasks because the budget-verification of small units probably does not take software-sided interdependencies into account, i.e. is not case-sensitive. However, a case-sensitive WCET analysis of greater units for budget-verification is assumed to be very complex due to the software's black-box character. WCET analysis has to take hardware-sided interdependencies into account too, like described in section 3.5. All issues require control over the WCET analysis. For that, the current methods using INCA and hardware traces, that are late in development because they call for a running system, are not suitable in general. A further possibility, compiling and integrating the new code into an already existing system, measuring its execution times and converting that results into that of the target system provides usable execution times only when the hardware is comparable, like described in section 3.6.1. However, one risks missing the WCET respectively the worst-case situation unless the code could be stimulated with certain input when using INCA or the debugger method. A static tool like aiT or the Tessy-approach seem to be more suitable.
The approach of code with constant execution times (see section 3.4.4) has advantages concerning timing-budgeting too. It is assumed to make timing-budgeting and timing-budget verification less difficult because the code's execution times are assumed to be less case-sensitive. So, a timing-budget can be specified for bigger units without the need for case-sensitive timing-budget verification. So, even dynamic WCET analysis methods like INCA measurements should provide reliable results, if the hardware-affects are in-



المنارة للاستشارات
www.manaraa.com

significant, because the black-box character of the software carries no weight under those circumstances. However, this approach might lead to higher execution times respectively resource requirements, see section 3.4.4.

### 3.6.3 Conclusion of Scenario 3

On the one hand, the benefit of SymTA/S is restricted if no information about a system is available, e.g. in very early development phase of a new project. SymTA/S analysis results depends on the precision of its input parameters. If these values are very imprecise, SymTA/S results are imprecise. That could lead to a misjudgment of a project. On the other hand, it provides a standardized format that allows storing, visualizing and the exchange of system configurations (proposals) between developers. SymTA/S provides the specification of several scenarios within one system. Each scenario can have its own parameter values, e.g. for WCETs, activation offsets, CAN-frame activation assumptions etc. This allows modeling several cases within one SymTA/S system instead of modeling multiple systems within SymTA/S. The sensitivity analysis helps to determine timing budgets that base on a certain system configuration. The design space exploration allows refining the system configuration if the WCETs become more precise. If no information is known, SymTA/S allows comparing several system configurations and WCETs estimation proposals. It also allows examining the effect when using another operating system or bus-system and so could help to take the right decision.
The verification if the timing budget is met depends on the concrete project. Several approaches are described which have to be selected depending on the project. Here, the sensitivity analysis might be necessary to refine the timing budget if it turns out that a certain assumption is not suitable. In an extreme case, the system configuration has to be changed supported by the SymTA/S' design space exploration.

## 3.7 Summary

This chapter describes the standard OSEK OS and one possible implementation used in automotive industry. OSEK OS specifies a real-time capable operating system. It does not consider deadlines in the proper meaning. Rather, it has a static priority based scheduler and implements the PCP in order to avoid deadlocks. Section 3.2.2 turns out that "real-time" in automotive industry depends on how the safety relevant the system is. In case of automatic transmission, a missed task cycle time in most cases is less critical than a missed deadline in the vehicle dynamics control or brake system.
The scheduling analysis tool SymTA/S calculates best- and worst-case response times of tasks, bus-frames or complicated end-to-end paths. The tool follows the so-called *compositional scheduling analysis*, see section 3.3.2. Local scheduling analysis is performed by certain libraries, e.g. for OSEK OS, CAN or FlexRay. SymTA/S eventually assumes pessimistic delays when leaving synchronized groups on end-to-end paths. So, the results



المنارة للاستشارات
www.manaraa.com

are safe but might be pessimistic in some cases, like described in sections 3.3.5 and 3.5.
The examination of scenario 1 clarifies that the available WCET analysis methods, INCA and processing of hardware-traces, are not suitable for system verification, see section 3.4. The reason is the black-box character of the software that does not allow determining and analysing the worst-case situation. Even using case-insensitive measured maximum execution times for scheduling analysis by SymTA/S does not qualify to provide a safe result because the worst-case situation and the WCETs are unknown. Proposals to get WCETs and safer WCRTs more easily are given. Analysis and modeling a system with SymTA/S is straightforward. The GUI is well suited for the intended purposes. It is a "one-click-analysis".
Scenario 2 is described in section 3.5. It suffers from the same problems like scenario 1, especially when extending an existing system with a further functionality. However, the handshake protocol-example shows the tool's greatest benefit. It visualizes the worst-case situation, e.g. of an end-to-end path within the system. These report files and Gantt-charts could be used for documentation purposes or as possibility to explain complicated issues. However, sampling effects that could lead to information loss should be marked more noticeable. Currently, these effects and pessimist delays that are determined by SymTA/S should be checked manually.
Scenario 3 is a theoretic consideration. Section 3.6 turns out that the precision of a timing budget highly depends on the given basis. If no information is available, determining a suitable timing budget is assumed to be very difficult. Precise execution time cross-estimations are possible only in a few cases. Sensitivity analysis and design space exploration could help to find a suitable system configuration within an iterative process. The precision of the SymTA/S model is assumed to increase with progressing project development. However, the use of both functionalities assumably becomes important when dealing with complex (synchronized) systems, e.g. systems using FlexRay.
The detailed listing of all bugs within SymTA/S is renounced because SymTA/S is a permanently improved tool and the bugs are not serious. It provides further functions concerning communication, e.g. traffic shaping. These functions are not needed in order to investigate the tool for the current purposes of ZF. As it turns out that the useful assignment of a timing tool like SymTA/S requires a lot of extra work, the examination of these functions is left for further work respectively when it is needed and can be understood on a concrete project.



المنارة للاستشارات
www.manaraa.com

# 4 Real-Time Analysis with chronSim

SymTA/S is a timing, respectively scheduling, analysis tool at an abstract level. It is intended to be used for system verification. Therefor, the input parameters like WCETs have to be provided. Currently, this is late in development process of ZF and the analysis results potentially are too pessimistic.
Therefor, an alternative tool is introduced in the following. The tool chronSim is developed by Inchron [29] and it is intended to *simulate* and *visualize* the timing behavior of a system. It is not intended to determine a potentially (pessimistic) worst-case situation, i.e. to verify the timing behaviour of a system. For this purpose, the tool chronVal is under development by Inchron. chronSim is able to simulate and visualize the timing behavior of single ECUs and distributed systems, comparable with SymTA/S use scenarios 1 and 2. In order to simulate the timing behavior of a system with chronSim, the system has to be modeled. In an early development phase, this can be done by files written in ANSI-C. These files contain the skeletal structure of the software, including branches, loops, function calls and synchronization elements like semaphores but they do not have to contain the original code. Thereby, the API of the target or a generic operating system can be used. The corresponding execution times of the software are specified by key words within these ANSI-C-files, like illustrated by figure 4.1. For a first simulation, execution

```
TASK(TASK_TT_5ms){
 DELAY(300, unit_us);
 Schedule();
 while(!finished()) {
   DELAY(100, unit_us);}
 exectime = 10*data_size;
 DELAY(exectime,unit_us);
 DELAY(gaussian(500, 10),unit_us);
 TerminateTask();
}
```

Figure 4.1: Example of a chronSim task model with specified execution times and operating system system calls (from chronSim presentation material)

time estimations could be used that are determined like described in sections 2.6.5 and 3.6. So, chronSim also can be used for a budgeting approach like SymTA/S too. It is



المنارة للاستشارات
www.manaraa.com

also possibly to specify a certain execution time probability distribution like the normal probability curve (see figure 4.1). However, using such a probability distribution makes it difficult to re-produce the simulation results.
In a later development phase, the original C-code can be used. Its execution on the target hardware running the target operating system can be simulated. Therefor, the C-code is cross-compiled considering specific compiler settings and the target hardware. So, the execution times of the code are determined by chronSim. A combination of original C-code and specified execution times is possible too. Inchron provides libraries for the compiler, the (real-time) operating system, the bus system and the processor, e.g. libraries for OSEK and FlexRay. It is possible to import the task model, i.e. scheduling strategy and task priorities, through OIL-files.
In both cases, the simulator executes each line of C- respectively assembler-code. It considers internal and external interdependencies and inputs. In order to get meaningful simulation results, appropriate stimuli, e.g. stress tests and suitable ISR activation scenarios, have to be provided. So, chronSim is not intended to determine the worst-case situation because the worst-case input is unknown in general due to the black-box character of the software. Maybe, it might make the timing behavior of a system more transparent because the simulation can be performed on single software parts and not only on the final software level after integration. If specifying WCETs, e.g. which are determined statically, the simulation by chronSim probably provides results that are comparable with that of SymTA/S. SymTA/S combines WCETs and calculates (eventually pessimistic) WCRTs. chronSim would also combine WCETs but does not guarantee to catch the worst-case combination which leads to the WCRT during simulation respectively it is unknown if this combination is caught.
All in all, chronSim moves the timing test of a system from currently integration phase to an earlier development phase. Therefor, a kind of **manual integration of the software** is necessary in order to create the system model within chronSim. It has to be investigated how much effort is necessary to do this. However, chronSim might help a software developer specifying timing information of his code and **exchanging information about the timing behavior of the code in an abstract way**. The timing behavior of the system might be understood better on this (abstract) code and possible bottlenecks might be detected. The effort to do this for the whole software has to be investigated further but is assumed to be very difficult due to the black-box character of the software. Some software parts, e.g. supplied software in tasks with a low cycle time, have a **black-box character**. Hence, an abstract and information hiding model to exchange chronSim-model information is desirable. Otherwise, if timing information about the supplied code is needed in an early development phase, it had to be requested from the supplier. In comparison with SymTA/S, chronSim takes the hardware more into account and allows simulating the code while considering the target hardware. Therefor, Inchron provides libraries for several processors which consider caches and pipeline. **These processor libraries base on benchmarks of reference code**. So, it has to be investigated if the accuracy of the processor model can be warranted for worst-case examinations. Further, some informa-



المنارة للاستشارات
www.manaraa.com

tion about the hardware possibly has to be requested by the hardware supplier like it is the case in section 2.6.4. Unfortunately, chronSim currently **does not handle C++**. It has to be investigated if the chronSim-model could be **integrated in the common version management** of the code.



المنارة للاستشارات
www.manaraa.com

# 5 Further Work

The examination of SymTA/S turns out that a usage of all tool's functionalities require further investigations. Most important seems to be a possibility to efficiently get safe and tight WCETs. The following points name several proposals to reach that:

- Examination of a programming style that provides constant execution times
- Examination of a static WCET analysis tool like aiT
- Adaption of the hardware-trace processing scripts
- Finishing the Tessy approach
- Examination of chronSim
- Development of a method to identify several WCET cases/combinations on a concrete project in order to determine the necessary effort to perform very precise timing analyses

Like described in section 3.3.7, a detailed investigation of the design space exploration internals would be necessary if critical system configuration decisions should be supported by the tool.



المنارة للاستشارات
www.manaraa.com

# 6 Conclusion

Chapter 2 turns out that complex hardware and software makes safe and precise WCET analysis difficult. Most available static WCET analysis tools support only a restricted set of microcontrollers. The tool aiT quickly provides first results. However, numerous annotations concerning the code and/or the hardware are necessary in order to get tight results. Under the current circumstances, black-box testing is not sufficient. It is supposed that the current measurement methods provide imprecise (worst-case) execution times. The Tessy approach, which provides a controlled dynamic WCET analysis, could not be finished due to missing information about the ECU, e.g. startup-code. However, WCET analysis that bases on the current measurement methods might be necessary for code that is supplied to ZF as black-box.
Chapter 3 describes the internals of SymTA/S and investigates which effort it provides in several use scenarios. **Scenario 1** (section 3.4) examines the analysis of an existing system. If SymTA/S detects potential timing problems, an analysis refinement would be necessary in order to verify or falsify that the problems can really occur. However, this is not manageable due to the black-box character of the software. One possibility to ease that problem is to produce code with constant execution times. At this point it is very important to note that SymTA/S only provides worst-case examinations that base on the given WCETs. It does not provide a probability of this worst-case situation. For a probability distribution of potentially problematic scheduling situations, all execution times and their possible combinations had to be known. This would imply knowledge of the worst-case situation. However, neither all possible execution times nor the worst-case situation are known due to the software's and hardware's complexity. Probability distributions of execution times that base on a representative driving situation are not sufficient because it is the indention to detect unknown special cases that do not occur during normal testing but possibly can occur.
**Scenario 2** clarifies that SymTA/S could be used to visualize complex end-to-end WCRTs. However, several effects are not displayed by SymTA/S at first sight. So, it has to be checked manually if some information could be lost, e.g. due to sampling effects. Section 3.5 describes an example system which is partly situated in implementation phase of the V-model. Because both ECUs are not synchronized, WCET examinations of not yet implemented code are sufficient to represent an approximate system behavior. At this point, not such precise and difficult to determine input parameters are necessary like in scenario 1. In this case, SymTA/S is supposed to have its greatest benefit. The report files and Gantt-charts that can be generated are suitable for documentation purposes.
**Scenario 3** (section 3.6) suffers from very imprecise raw data. Several assumptions have



المنارة للاستشارات
www.manaraa.com

to be made. Time budgeting could be supported by the sensitivity analysis. The optimization functionality is assumed to make sense when at least a few precise parameters are known. The black-box character of the software makes it difficult to specify realistic timing-budgets. Software with constant execution times assumably makes the use of SymTA/S for this purpose more easily.
Chapter 4 introduces a further timing analysis tool which is intended to simulate the timing behavior of a given system instead of verifying it. All in all, the use of SymTA/S is limited to a few situations that do not exhaust the capabilities of the tool under the current conditions.



المنارة للاستشارات
www.manaraa.com

# Glossary

ABS ............... Anti-lock Brake System
ANSI .............. American National Standards Institute
AOMF ............. Absolute Object Module Format
ASCII ............. American Standard Code for Information Interchange
AUTOSAR ......... Automotive Open System Architecture
BCC ............... Basic Conformance Class
BCET ............. Best-Case Execution Time
CAN .............. Controller Area Network
CC ................ Conformance Class
CFG ............... Control Flow Graph
COFF .............. Common Object File Format
CTE ............... Classification Tree Editor
CPU ............... Central Processing Unit
CRL ............... Control-Flow Representation Language
CSA ............... Context Save Area
EABI .............. Embedded Application Binary Interface
ECC ............... Extended Conformance Class
ECU ............... Electronic Control Unit
ELF ............... Executable and Linking Format
ETAS .............. Entwicklungs-, Test- und Applikationssysteme
FIFO .............. First In First Out
GPL ............... GNU General Public License
GUI ............... Graphical User Interface
HTML ............. Hypertext Markup Language
ILP ................ Integer Linear Programming
IPET .............. Implicit Path Enumeration Technique
ISO ................ International Organization for Standardization
ISR ................ Interrupt Service Routine
ITA ................ Instruction Timing Addition
LIN ............... Local Interconnect Network
LMI ............... Local Memory Bus Interface
LOC ............... Lines Of Code
LRU ............... Last Recently Used
OCDS ............. On-Chip Debug System



المنارة للاستشارات
www.manaraa.com

OIL ................ OSEK Implementation Language
OTAWA ............ Open Tool for Adaptive WCET Analyses
OSEK ............. Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
PCP ................ Priority Ceiling Protocol
PMI ................ Program Memory Interface
RAM .............. Random Access Memory
RTA-OSEK ........ Real-Time Architect - OSEK
ROM .............. Read Only Memory
RMS .............. Rate-Monotonic Scheduling
SWEET ............ Swedish Execution Time Tool
TEE ................ Tessy Environment Editor
TIMMO ........... Timing Model
TLB ................ Translation Look-Aside Buffer
WCET ............. Worst-Case Execution Time
XML .............. Extensible Markup Language



المنارة للاستشارات
www.manaraa.com

# Bibliography


[1] **A Worst-Case Execution-Time Analysis Tool Prototype for Embedded Real-Time Systems**, Jakob Engblom, Andreas Ermedahl, Friedhelm Stappert, August 2001, PDF
http://c-lab.de/fileadmin/redactors/data/Publication_DB/DB/Engblom__00/rttools01.pdf

[2] **Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints**, Patrick Cousot, Radhia Cousot, 1977, PDF
http://groups.csail.mit.edu/pag/6.883/readings/p238-cousot.pdf

[3] **Abstract Interpretation in a Nutshell**, Bruno Blanchet et. al, November 2007
www.astree.ens.fr/IntroAbsInt.html

[4] **aiT WCET Analyzers: Requenst Evaluation**, aiT free trail version, 2007
www.absint.com/ait/trial.htm

[5] **aiT: Worst Case Execution Time Analyzers**, aiT tool homepage, November 2007
www.absint.com/ait/

[6] **aiT white paper: Worst-Case Execution Time Prediction by Static Program Analysis**, Reinhold Heckmann, Christian Ferdinand, PDF
www.absint.com/aiT_WCET.pdf

[7] **AOMF with Keil C51 extensions as input to Bound-T**, Technical Note, September 2007, PDF
www.tidorum.fi/bound-t/tech_notes/tn-aomf.pdf

[8] **Applying Static WCET Analysis to Automotive Communication Software**, Susanny Byhlin, Andreas Ermedahl, 2005, PDF
www.mrtc.mdh.se/publications/0974.pdf

[9] **AUTOSAR**, homepage, October 2007
www.autosar.org

[10] **Bound-T Application Nodes**
www.tidorum.fi/bound-t/app_notes




المنارة للاستشارات
www.manaraa.com

[11] **Bound-T User Manual**, October 2007, PDF
www.tidorum.fi/bound-t/user-manual.pdf

[12] **C++test**, homepage, November 2007
www.parasoft.com/jsp/products/home.jsp?product=CppTest

[13] **Calculating Controller Area Network (CAN) Message Response Times**, K. Tindell, A. Burns, A. Wellings, 1995
http://citeseer.ist.psu.edu/291115.html

[14] **CAN**, homepage, October 2007
www.can-cia.org

[15] **Chalmers University of Technology: Timing analysis methods**, project homepage, November 2007
www.ce.chalmers.se/research/group/hpcag/project/wcet.html

[16] **Chronos**, homepage, November 2007
www.comp.nus.edu.sg/%7Erpembed/chronos/

[17] **Chronos: a Timing Analyzer for Embedded Software**, Xianfeng Li and Yun Liang and Tulika Mitra and Abhik Roychoudhury, 2005, PDF
www.comp.nus.edu.sg/%7Erpembed/chronos/chronos_paper.pdf

[18] **Cinderella**, project homepage, November 2007
www.princeton.edu/%7Eyauli/cinderella-3.0/

[19] **Echtzeit-Systeme**, lecture note, R. Baumgartl, 2006, PDF
http://rtg.informatik.tu-chemnitz.de/docs/ezs/ezs-skript-06.pdf

[20] **ETAS - INCA Software Products**, homepage, December 2007
www.etas.com/de/products/inca_software_products.php

[21] **EvoComp**, brief survey over unit testing, November 2007
www.evocomp.de/softwareentwicklung/unit-tests/unittests.html

[22] **Experiences from Applying WCET Analysis in Industrial Settings**, Jan Gustafsson, Andreas Ermedahl, 2005, PDF
www.mrtc.mdh.se/publications/1271.pdf

[23] **Experiences from Industrial WCET Analysis Case Studies**, Andreas Ermedahl, Jan Gustafsson, Björn Lisper, 2005, PDF
www.mrtc.mdh.se/publications/0966.pdf

[24] **Fifth Iinternational Workshop on Worst-Case Execution Time Analysis**, Reinhard Wilhelm, 2005, PDF
http://artist.cs.uni-sb.de/WCET05/Papers/WCET2005Proceedings.pdf



المنارة للاستشارات
www.manaraa.com

[25] **FlexRay**, homepage, October 2007
www.flexray.com

[26] **Heptane - Static WCET Analyzer**, homepage, November 2007
www.irisa.fr/aces/work/heptane-demo/heptane.html

[27] **HSE Free OSEK**, project homepage, January 2008
www.hs-esslingen.de/%7Efrruit00/

[28] **ILOG CPLEX**, homepage, November 2007
www.ilog.com/products/cplex

[29] **Inchron**, homepage, February 2008
www.inchron.com

[30] **LIN**, homepage, October 2007
www.lin-subbus.org

[31] **lp_solve reference guide (5.5.0.10)**, homepage, November 2007
http://lpsolve.sourceforge.net/5.5

[32] **Modeling out-of-order processors for WCET analysis**, Xianfeng Li, Abhik Roychoudhury, Tulika Mitra, PDF
www.comp.nus.edu.sg/%7Eabhik/pdf/rts-jnl06.pdf

[33] **OTAWA**, project homepage, November 2007
www.otawa.fr

[34] **OSEK OS specification**, v.2.2.3, February 2005, PDF
http://portal.osek-vdx.org/files/pdf/specs/os223.pdf

[35] **OSEK VDX**, homepage, January 2008
www.osek-vdx.org

[36] **RapitTime White Paper**, September 2006, PDF
www.rapitasystems.com/system/files/RapiTime_WhitePaper.pdf

[37] **Rational Test RealTime**, homepage, November 2007
www.ibm.com/software/awdtools/test/realtime/sysreq

[38] **Real-Time Analysis as a Quality Feature: Automotive Use-Cases and Applications**, Kai Richter, February 2006, PDF
www.symtavision.com/downloads/Ric06-RealTAnalyQualiFeatu.pdf

[39] **Scheduling Analysis Integration for Heterogeneous Multiprocessor SoC**, Kai Richter et. al., December 2003



www.manaraa.com

[40] **SimpleScalar Tools**, homepage, November 2007
http://pages.cs.wisc.edu/%7Emscalar/simplescalar.html

[41] **Software Testing FAQ**, Unit Test Tools, November 2007
www.testingfaqs.org/t-unit

[42] **Swedish Execution Time Tool**, project homepage, November 2007
www.mrtc.mdh.se/projects/wcet/sweet.html

[43] **Symta Vision**, homepage, October 2007
www.symtavision.com

[44] **SymTA/S Manual Introduction and Theory**, Version 1.3, October 2007

[45] **System Level Performance Analysis - the SymTA/S approach**, Rafik Henai et. al., September 2004, PDF
www.symtavision.com/downloads/System_Level_Performance_Analysis-the_SymTA-S_Approach.pdf

[46] **Systematic Testing**, homepage, November 2007
www.systematic-testing.com

[47] **TASKING TriCore VX development tools**, November 2007
www.tasking.com/products/32_bit/tricore/

[48] **Tessy application note - Return Value Sequences for Stubs**, Frank Buechner, February 2002, PDF
www.hitex.de/pdf/application_notes/AN-TESSY-0101.pdf

[49] **Tessy application note - Testing State Machines**, Frank Buechner, August 2006, PDF
www.hitex.de/pdf/application_notes/AN-TESSY-0105.pdf

[50] **Tessy White Paper - Unit Test of Embedded Software**, Frank Buechner, December 2004

[51] **Testwell**, homepage, November 2007
www.testwell.fi

[52] **The SimpleScalar Tool Set, Version 2.0**, Doug Burger, Todd M. Austin, June 1997, PDF
www.eecs.umich.edu/%7Etaustin/papers/UWTR97-simple.pdf

[53] **The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools**, Reinhard Wilhelm et. al, January 2007, PDF
www.cs.fsu.edu/%7Ewhalley/papers/tecs07.pdf




المنارة للاستشارات
www.manaraa.com

[54] **The Worst-Case Execution Time Tool Challenge 2006**, Lili Tan, 2006
www.absint.com/ait/WCET_Tool_Challenge_2006_Final_Report.pdf

[55] **Tidorum Ltd**, homepage, 2007
www.tidorum.fi

[56] **Timing analysis of embedded software for speculative processors**, Tulika Mitra, Abhik Roychoudhury, Xianfeng Li, 2002, PDF
www.comp.nus.edu.sg/%7Etulika/isss02.pdf

[57] **TIMMO**, project homepage, October 2007
www.timmo.org

[58] **Using the classification tree method**, November 2007
www.embedded.com/story/OEG20020613S0019

[59] **V-Model**, Overview, October 2007
www.informatik.uni-bremen.de/gdpa/vmodel/i-gral.htm

[60] **VectorCAST**, homepage, November 2007
www.vectors.com/product.htm

[61] **Verfahren zur Bestimmung der Worst Case Execution Time (WCET)**, A. Kaiser, June 2005, PDF
http://www4.informatik.uni-erlangen.de/%7Ewawi/AKES/Handout-AKES-WCET.pdf

[62] **Vienna Real-Time Systems Group**, homepage, November 2007
www.vmars.tuwien.ac.at

[63] **Worst Case Execution Time Analysis of Object-Oriented Programs**, J. Gustafsson, 2002, PDF
http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7841/21579/01000038.pdf

[64] **ZF Friedrichshafen AG**, homepage, October 2007
www.zf.com



المنارة للاستشارات
www.manaraa.com

# Declaration of Authorship

I hereby declare that the whole of this diploma thesis is my own work, except where explicitly stated otherwise in the text or in the bibliography. This work is submitted to Chemnitz University of Technology as a requirement for being awarded a diploma in Computer Science ("Diplom-Informatik"). I declare that it has not been submitted in whole, or in part, for any other degree.

Friedrichshafen, March 31, 2008

_______________________

Martin Däumler



المنارة للاستشارات
www.manaraa.com