Integrating System-Level and Code-Level Timing Analysis for Dependable System Development

C Ferdinand, R Heckmann, M Jersak, F. Martin, K Richter

Integrating System-Level and Code-Level Timing Analysis for Dependable System Development

C. Ferdinand[1], R. Heckmann[1], M. Jersak[2], F. Martin[1], K. Richter[2]

1: AbsInt Angewandte Informatik GmbH, Science Park 1, D-66123 Saarbrücken, Germany

2: Symtavision GmbH, Frankfurter Straße 3b, D-38122 Braunschweig, Germany

Abstract: Developers of safety-critical real-time systems have to ensure that their systems react within given time bounds. Sophisticated tools for timing analysis at the code-level, controller-level and networked system-level are becoming state-of-the-art for efficient timing verification in light of ever increasing system complexity. This trend is exemplified by two tools: AbsInt's timing analyzer aiT, which can determine safe upper bounds for the execution times (WCETs) of non-interrupted tasks, and Symtavision's SymTA/S tool, which computes the worst-case response times (WCRTs) of an entire system from the task WCETs and information about possible interrupts and their priorities. The two tools thus complement each other in an ideal way. They have recently been coupled to further increase their utility. Starting from a system model, a designer can now seamlessly perform timing budgeting, performance optimization and timing verification, considering both the code of individual functions, as well as function and sub-system integration. The paper explains and exemplifies various use cases and tool flows.

Keywords: Schedulability analysis, timing analysis, worst-case execution time, worst-case response time

1. Introduction

Developers of safety-critical real-time systems have to ensure that their systems react within given time bounds. Tests and measurements cannot guarantee that this holds in every possible situation, but tools for static program analysis can obtain results valid for all possible system runs and inputs, even before the first real prototypes are available. Examples for such tools are AbsInt's timing analyzer aiT, which can determine safe upper bounds for the execution times (WCETs) of non-interrupted tasks, and Symtavision's SymTA/S tool, which computes the worst-case response times (WCRTs) of an entire system from the task WCETs and information about possible interrupts and their priorities.

The focus of aiT on single tasks and the focus of SymTA/S on the interplay of several tasks complement each other in an ideal way. Coupling these tools therefore highly increases their utility. The system developer creates a system model in SymTA/S, consisting of a task graph, information on task scheduling (priorities, time slots, etc.), and information on task activation (time tables, interrupts, etc.). To determine the WCRTs of the tasks with possible interrupts, the WCETs of the non-interrupted tasks are required. To obtain these WCETs, SymTA/S sends requests to aiT. Then aiT asks the developer for necessary information on hardware configuration and executables, determines the requested WCETs, and sends them back to SymTA/S. To increase the user comfort, the communication is based on cookies in which the configuration information is stored so that this information need not be input again when further WCET calculations are performed. These cookies are organized hierarchically: general information valid for the entire system, information about all parts of the system running on a specific CPU, information for runnables (i.e., atomic pieces of software), and modes (specific control-flow paths through a runnable).

The coupling of SymTA/S with aiT allows SymTA/S to determine end-to-end timings in an early development phase, with automatic identification of problematic system configurations and automatic system optimization as a next step. In the EU project INTEREST (Integrating European Embedded Systems Tools) the tool coupling is extended towards modelling tools such as ASCET and SCADE to support the entire development process from the model till the implementation with formally verified timing behavior.

In the following, we present the tools, the experimental integration, preliminary results and plans for further tool integration.

2. The WCET Analyzer aiT

aiT is AbsInt's timing analyzer, which can find upper bounds for the worst-case execution times (WCETs) of sequential tasks. For a precise computation of the WCET, aiT operates on the executable.

The graphical user interface of aiT (see Figure 1)

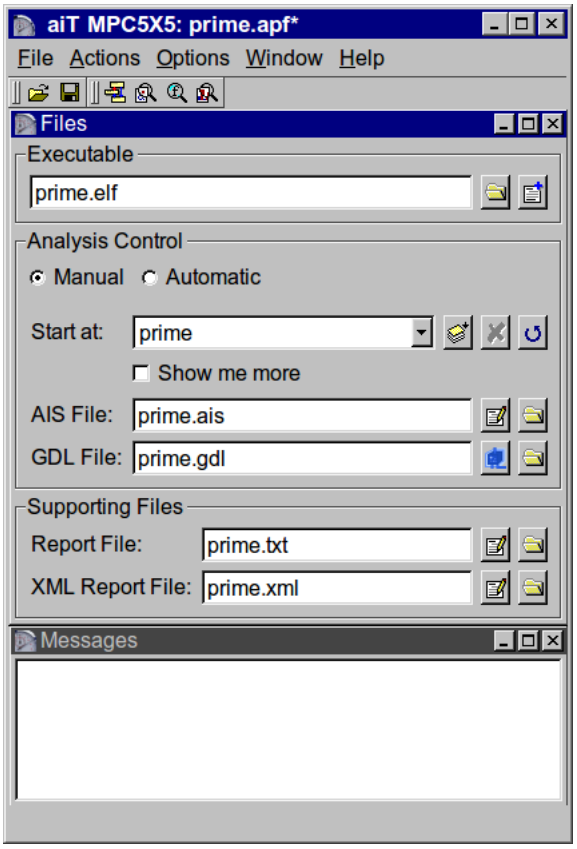

Figure 1: Graphical user interface of aiT

offers ways to specify the memory architecture of the target, the location of source files, the name of the executable, the name of a separate parameter file called `.ais` file, the name of the report files to be written, etc., and the start point of the analysis (a routine name or an address). All this information can be stored in a project file (`.apf` file). The `.ais` file may contain the clock rate of the target processor, upper bounds for the iteration numbers of loops, possible targets of computed calls, etc. The analyses of aiT are mainly based on the executable. If available, aiT can also read the source files for further information. The association between addresses in the executable and positions in the source files is obtained from the debug information in the executable.

aiT can be started in three ways:

1. When started for interactive usage, the GUI depicted in Figure 1 is opened. The fields in the GUI can be filled with appropriate values, which may be stored in a project file (`.apf` file). Alternatively, an existing project file can be loaded.
2. aiT can also be started in simple batch mode with a project file. In this case, the project file is loaded and then aiT behaves as if the "WCET analysis" button had been clicked.
3. aiT can be started in batch mode with a control file. A control file contains the name of a project file and specifies various analysis tasks, which are performed automatically in this case. Control files can also be used interactively if the GUI is switched from Manual to Automatic mode (see Figure 1).

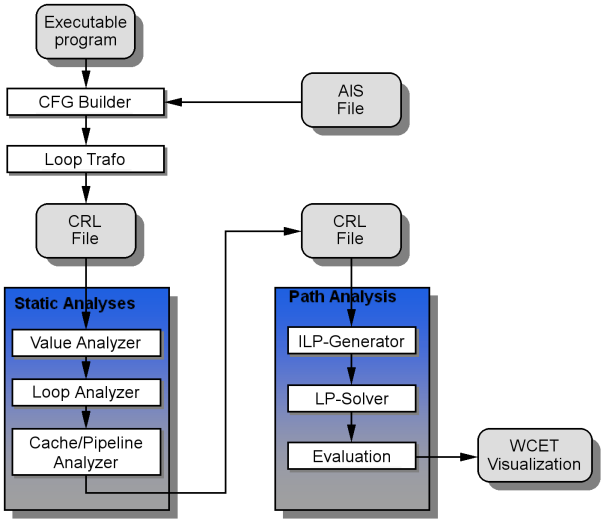

Figure 2: Phases of WCET computation

aiT determines the WCET of a given task in several phases [2] (see Figure 2). In the first step a *decoder* reads the executable and reconstructs the control flow [9]. Then, *value analysis* determines lower and upper bounds for the values in the processor registers for every program point and execution context, which lead to bounds for the addresses of memory accesses (important for cache analysis and if memory areas with different access times exist). Value analysis can also determine that certain conditions always evaluate to true or always evaluate to false. As consequence, certain paths controlled by such conditions are never executed. Thus value analysis can detect and mark some unreachable code.

WCET analysis requires that upper bounds for the iteration numbers of all loops be known. aiT tries to determine the number of loop iterations by *loop bound analysis* [4], but succeeds in doing so for simple loops only. Bounds for the remaining loops must be provided as specifications in the AIS file or annotations in the C source.

If the target processor has caches, an optional *cache analysis* follows, which classifies the accesses to main memory into hits, misses, or accesses of unknown nature. *Pipeline analysis* models the pipeline behavior to determine execution times for sequential flows (basic blocks) of instructions as done in [8]. It takes into account the current

pipeline state(s), in particular resource occupancies, contents of prefetch queues, grouping of instructions, and classification of memory references by cache analysis. The result is an execution time for each basic block in each distinguished execution context.

Using this information, *path analysis* determines a safe estimate of the WCET. The program's control flow is modeled by an integer linear program [6, 10] so that the solution to the objective function is the predicted worst-case execution time for the input program.

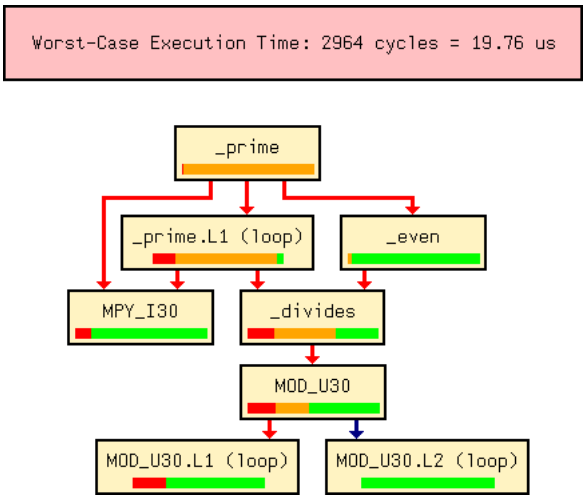

Figure 3: Call graph with WCET result

After a successful analysis, aiT reports its results in several ways:

1. aiT can produce a graphical output showing the call graph of the analyzed part of the application, showing the routines and their calling relationships (see Figure 3). The routine boxes can be opened to show their control-flow graphs with WCET results for basic blocks. Technically, aiT writes a description of these graphs in a GDL file, which can be visualized by AbsInt's graph browser aiSee.
2. aiT can write a text report meant to be human readable, and a more formal XML report. These reports contain detailed results for all analyzed routines in all calling contexts, including specific results for the first few iterations of loops vs. a result for the remaining iterations.

3. Scheduling Analysis with SymTA/S

SymTA/S [5] is Symtavision's tool for timing and scheduling analysis and optimization for controllers, networks and entire systems. SymTA/S computes

- worst-case response times (WCRTs) of tasks
- worst-case end-to-end communication delays

taking into account all relevant WCETs and information about RTOS scheduling, bus arbitration, possible interrupts and their priorities.

Scheduling analysis is a systematic approach that automatically finds and evaluates critical timing situations resulting from function and system integration. Such *corner case identification* is the opposite of traditional test-based methods: instead of massive testing to try to find all corner cases, scheduling analysis systematically constructs scenarios leading to worst-case timing.

SymTA/S combines decades of research (started with Liu's and Layland's 1973 paper on Rate-Monotonic scheduling [7]) into an industry-strength tool. SymTA/S provides detailed analysis results that are comprehensively visualized and thus easy to understand.

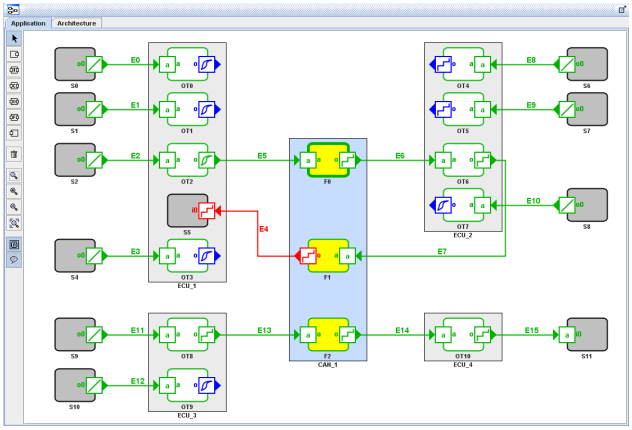

Figure 4: Graphical user interface of SymTA/S (application view)

The graphical user interface of SymTA/S (see Figure 4) offers ways to specify a system architecture, select scheduling on controllers and arbitration on buses, map functions to controllers and communication to busses, and to describe dataflow, activation conditions, deadlines and other timing constraints.

The analysis results are displayed in a variety of ways. The most powerful and easy to understand are *Gantt-Charts* that visualize to the designer why and under which conditions deadlines can be violated (see Figure 5).

There are two main use cases for SymTA/S:

1. *timing design / budgeting* during early design stages and
2. *timing verification* during later design stages.

The added value for verification is exemplified in Figure 6: the upper part of the diagram displays a typical timing trace, showing a response time of

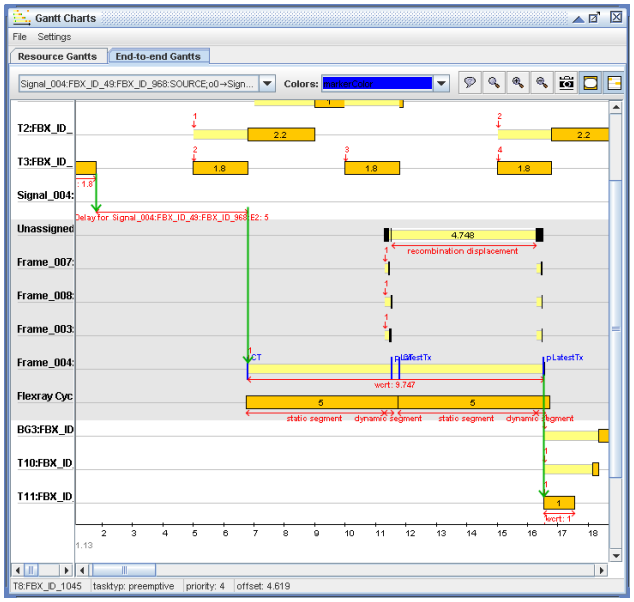

Figure 5: Graphical display of worst-case timing along a datapath (end-to-end Gantt chart)

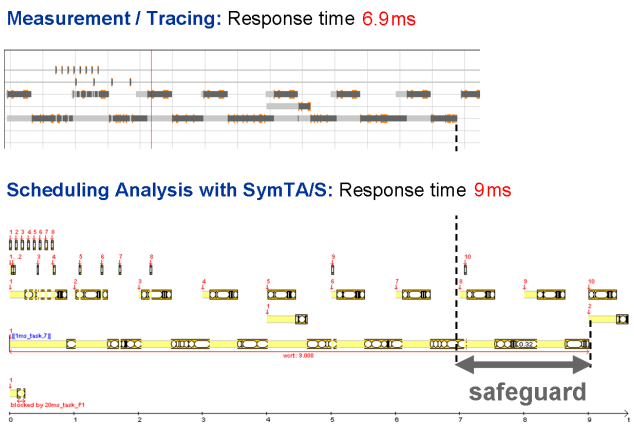

Figure 6: Safeguarding timing using scheduling analysis (as compared to measurement)

6.9 ms for a task executed every 10 ms – well below the 10 ms deadline. In the lower part, SymTA/S scheduling analysis has constructed a worst-case schedule leading to a WCRT of 9 ms for the 10 ms task – still below the deadline, but much closer. The key message is that no other schedule will produce a longer WCRT. SymTA/S thus safeguards against deadline violations resulting from worst-case schedules. Furthermore, the Gantt-display enables the designer to check the reasoning of SymTA/S and thus to see if some important information has been omitted in the model.

At the same time, SymTA/S allows to perform *what-if* scheduling analysis during system design. It is easily possible to estimate how additional functions and their scheduling will influence overall system timing, and whether deadlines can be safely met for a specific design alternative. As a result, timing budgets for individual functions of sub-systems can be derived early on, and given to a designer as part of a requirements specification.

SymTA/S supports timing design through productivity plugins, which can be integrated with the scheduling analysis engine.

1. *Design-space exploration* allows designers to automatically evaluate the strengths and weaknesses of alternative designs with respect to timing and performance.
2. *Sensitivity analysis* allows users to automatically determine the amount of extra load (e.g., caused by additional functions) permissible without violating deadlines.

4. XML Timing Cookies (XTC)

Timing analysis is a novel domain, and the requirements for coupling code-level and system-level tools were not suitably covered by any existing exchange format. Therefore, the concept of "Timing Cookies" has been developed in the INTEREST project, and an "XML Timing Cookies" exchange format (XTC) has been defined.

The main idea behind "Timing Cookies" stems from two observations:

1. The envisioned flow between SymTA/S and aiT is essentially cyclic, suggesting a request-response mechanism.
2. Each tool requires a (potentially large) set of data about the system under design, but the intersection of these two sets is small.

"Timing Cookies" have been introduced to avoid the duplication of the sophisticated user-interfaces available in each tool. The concept of "Timing Cookies" allows to keep entering the required information in the appropriate place, and to store the information for the next round of communication between the tools. This is similar to repeatedly visiting a web site that requires certain user information. Such information is typically stored in a cookie and retrieved when the user visits the site again, hence the name "Timing Cookie".

An XML Timing Cookie consists of two main parts:

1. One common section that describes the analysis request when the cookie is sent from SymTA/S to aiT, and additionally holds the response to that request when the cookie is returned from aiT to SymTA/S.
2. A cookie section per communicating tool to hold each tool's local information required for servicing a request and for putting the response in its appropriate context.

This partitioning has the following advantages:

- Key information is stored in one place, the common section.
- Tool-specific information is stored in a private cookie section. It is only modified by the respective tool; other tools leave this information untouched.

XTC is defined as an XML schema. It is organized hierarchically since some information can be reused for different analyses. For instance, a CPU configuration can be reused for different runnables sharing the same CPU. The information in the common section is structured in four blocks:

1. general
2. CPU
3. runnable (i.e., an atomic piece of software[1])
4. mode (a specific control-flow path through the runnable).

Structuring of the cookie section has to follow the top-level structuring of the common section. XML name spaces are used to avoid ambiguities during parsing. The schema definitions are available upon request.

5. The Interaction between SymTA/S and aiT

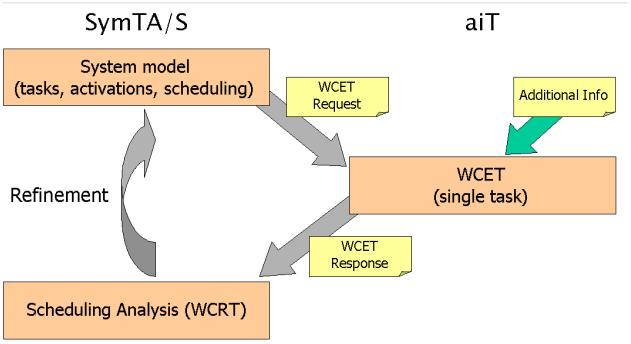

Figure 7: Flow of requests and responses

SymTA/S and aiT communicate by exchanging XML Timing Cookies (XTC, see section 4). From a system model, SymTA/S launches a request for WCET information for specific pieces of code (see Figure 7). This request is tagged with a unique ID and sent to aiT in an XTC. If necessary, aiT queries the user for all missing information required to service the request. For the first request issued for a system model, this typically includes the type of processor, the location of the executable code, the starting point of the analysis etc. When aiT answers the request by sending an XTC with a response back to SymTA/S, it stores this information in the private aiT-part of the cookie. This aiT-specific information will be included in subsequent requests so that aiT can use the information already gathered without the need to ask the user again.

[1] AUTOSAR [1] terminology has been adopted

6. Application Example

The example in this section has been adapted from an aeroplane controller. It illustrates the application of the described methodology and tools using a real-world example.

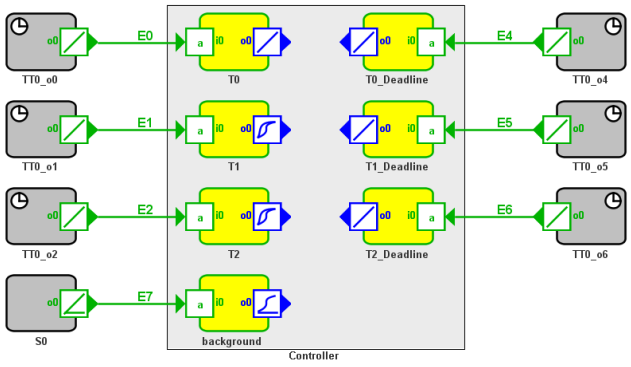

Figure 8: System example illustrating WCET analysis and WCRT analysis

The system is initially modelled in SymTA/S (see Figure 8). It consists of 3 main tasks[2] T0, T1, and T2, one background task, and several interrupt service routines (ISRs), most notably three ISRs performing deadline monitoring. A deadline overrun indicates a serious problem and must not occur under normal conditions. Therefore, WCET analysis and WCRT analysis are used during system design to validate the timeliness under all relevant conditions.

The three main tasks are triggered periodically with different periods and offsets. Such activation can be modelled using a time-table. We shall assume the following values:

task	period	offset	deadline	priority
T0	1000 us	50 us	200 us	high
T1	2000 us	270 us	1400 us	mid
T2	4000 us	1000 us	2800 us	low

A deadline is measured from the time of activation of the corresponding task.

Task T1 executes one major function $T1_{F1}$. The designer would like to add a second function $T1_{F2}$. However, this is only possible if T1 does not violate its deadline even if both $T1_{F1}$ and $T1_{F2}$ are executed. The other tasks must also not violate their deadlines.

[2] Note that *tasks* are sometimes referred to as *processes*.

In the first step, WCETs are obtained for T0, T2, $T1_{F1}$ and $T1_{F2}$. For this purpose, aiT is polled using an XTC *request* file. A simple example looks as follows:

```
<requestResponseCommunication>
  <common>
    <CPU unit="MHz" speed="10.0"
         name="Controller" id="ID_0">
      <runnable  name="T0" id="ID_1">
       <mode id="ID_2">
        <description>T0</description>
        <request type="WCET-Analysis"/>
       </mode>
      </runnable>
    </CPU>
  </common>
</requestResponseCommunication>
```

Since there are different instances of aiT for different target architectures, the cookie is not sent directly to aiT, but to an aiT driver program that asks for the target architecture unless this information is already contained in the cookie, and then calls the appropriate aiT instance. The aiT driver also translates the information in the cookie into the formats readable by aiT (project file, AIS parameter file, and control file).

If sufficient information for WCET analysis is already available, aiT can be run in batch mode. If not, it can be used interactively for entering the missing information, e.g., cache and memory configuration of the target processor, the path name of the executable, targets of computed calls and branches (unless found by static program analysis), loop bounds and recursion bounds (unless found by static program analysis), and declaration of volatile memory areas.

The information collected in this way is stored by aiT in its proprietary formats (project file, AIS parameter file, and control file) and returned to the aiT driver together with the results of the WCET analysis. The driver puts all this information into an XTC response file that is returned to SymTA/S. The WCETs are contained in the common part of the cookie for use by SymTA/S. The aiT-specific information is stored in the aiT-specific part of the cookie. This part can be copied by SymTA/S into the next XTC request file so that the information provided during the previous interactive aiT session need not be entered by the user again.

A simple example for the common part of an XTC *response* file looks as follows:

```
<requestResponseCommunication>
  <common>
    <CPU unit="MHz" speed="10.0"
         name="Controller" id="ID_0">
      <runnable  name="T0" id="ID_1">
       <mode id="ID_2">
        <description>T0</description>
        <request type="WCET-Analysis"/>
        <response type="WCET-Analysis">
         <WCET unit="us" value="180.0"/>
        </response>
       </mode>
      </runnable>
    </CPU>
  </common>
</requestResponseCommunication>
```

Let us assume that aiT returned the following values:

task	WCET
T0	180 us
$T1_{F1}$	1100 us
$T1_{F2}$	100 us
T2	300 us

Scheduling analysis can now be performed. Considering also the execution time of ISRs (not further discussed here but obtained using the same process), the resulting worst-case schedule and WCRT for T1 as obtained by SymTA/S is displayed in Figure 9.

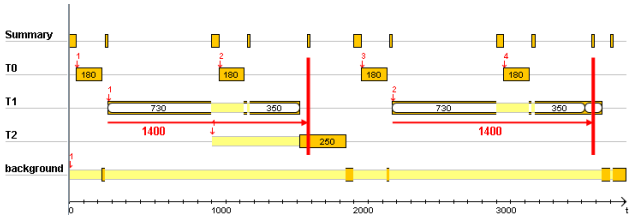

Figure 9: Bad schedule of example system missing the deadline for task T1 if both $T1_{F1}$ and $T1_{F2}$ are executed (2nd execution of T1)

As can be seen in Figure 9, the first execution of T1 where only $T1_{F1}$ is executed meets its deadline (left tall vertical bar). However, when also executing $T1_{F2}$ (second execution of T1), the deadline (right vertical bar) is missed.

The designer can now perform targeted timing optimization to resolve this problem. There are several options:

1. optimize the code of $T1_{F2}$ to reduce its WCET such that the deadline will be met;
2. optimize the assignment of offsets;
3. delay the deadline (if this is possible);
4. optimize the code of some other function to reduce its WCET;
5. some combination of these remedies.

Of the remedies, the first three are probably the most cost efficient, since they do not re-

quire re-certification of already certified functions. SymTA/S sensitivity analysis can be used to calculate by which amount each parameter would have to be changed.

Let us assume that code improvement of $T1_{F2}$ reduces its WCET to 50 us (verified by aiT). Sensitivity analysis tells us that this is not good enough, but there is an opportunity to start T1 earlier by up to 5 us. Combined, these two remedies mean that the deadline can be safely satisfied, with T0 maintaining a reserve of 15 us, and T1 maintaining a reserve of 5 us until its deadline (see Figure 10).

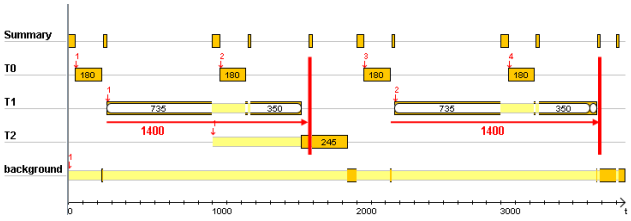

Figure 10: Fixed schedule of example system leading to all deadlines satisfied, even if both $T1_{F1}$ and $T1_{F2}$ are executed

7. Planned Extensions

Two extensions are planned for the near future:

1. Extending the XTC mechanism to also calculate stack usage. For many target architectures, the aiT tool can already determine the maximum stack usage of a task.
2. Integrating aiT and SymTA/S in larger tool flows.

For the latter, integration with ESTEREL's SCADE tool is well under way [3]. A second integration is being developed with ETAS's ASCET and INTECRIO. The goal is to provide seamless integration of timing analysis on all levels with established flows – from modelling/simulation to code-generation and code-optimization.

8. Conclusion

The code-level timing analysis tool aiT and the system-level timing analysis tool SymTA/S have been coupled to provide software and system designers with an efficient way to verify timing properties of ECU software.

The tool coupling allows you to

- determine upper bounds for the execution times (WCETs) of tasks using AbsInt's aiT;
- compute worst-case response times (WCRTs) of functions and tasks considering integration and scheduling using Symtavision's SymTA/S;
- trigger AbsInt's timing analyses at the code level from the SymTA/S GUI;
- import safe analysis results from aiT into SymTA/S in a fully automatic way.

XML Timing Cookies (XTC) provide for a user-friendly, open, and efficient tool integration. SymTA/S communicates with aiT via XTC, sending analysis requests and receiving responses.

9. Acknowledgement

Collaboration between AbsInt GmbH and Symtavision GmbH has been supported by the FP6 STREP project INTEREST (INTEgrating euRopean Embedded Systems Tools).

10. References

[1] The AUTOSAR Development Partnership. Automotive Open System Architecture (AUTOSAR). URL: http://www.autosar.org, 2003.

[2] Christian Ferdinand, Reinhold Heckmann, Marc Langenbach, Florian Martin, Michael Schmidt, Henrik Theiling, Stephan Thesing, and Reinhard Wilhelm. Reliable and precise WCET determination for a real-life processor. In *Proceedings of EMSOFT 2001, First Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 469–485. Springer-Verlag, 2001.

[3] Christian Ferdinand, Reinhold Heckmann, Thierry Le Sergent, Daniel Lopes, Bruno Martin, Xavier Fornari, and Florian Martin. Combining a high-level design tool for safety-critical systems with a tool for WCET analysis on executables. In *4th European Congress ERTS Embedded Real Time Software, Toulouse, France*, January 2008.

[4] Christian Ferdinand, Florian Martin, Christoph Cullmann, Marc Schlickling, Ingmar Stein, Stephan Thesing, and Reinhold Heckmann. New developments in WCET analysis. In Thomas Reps, Mooly Sagiv, and Jörg Bauer, editors, *Program Analysis and Compilation, Theory and Practice*, volume 4444 of *Lecture Notes in Computer Science*, pages 12–52. Springer-Verlag, 2007.

[5] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis – the SymTA/S approach. *IEEE Proceedings on Computers and Digital Techniques*, 152(2), March 2005.

[6] Yau-Tsun Steven Li and Sharad Malik. Performance Analysis of Embedded Software Using Implicit Path Enumeration. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, 1995.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[8] Jörn Schneider and Christian Ferdinand. Pipeline Behavior Prediction for Superscalar Processors by Abstract Interpretation. In *Proceedings of the ACM*

SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems, volume 34, pages 35–44, May 1999.

[9] Henrik Theiling. Extracting Safe and Precise Control Flow from Binaries. In *Proceedings of the 7th Conference on Real-Time Computing Systems and Applications*, Cheju Island, South Korea, 2000.

[10] Henrik Theiling and Christian Ferdinand. Combining abstract interpretation and ILP for microarchitecture modelling and program path analysis. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 144–153, Madrid, Spain, December 1998.

11. Glossary

AIS:	AbsInt's Specification language for user annotations
aiT:	AbsInt's Timing (and stack) analyzer
CPU:	Central Processing Unit
GDL:	Graph Description Language
GUI:	Graphical User Interface
ISR:	Interrupt Service Routine
RTOS:	Real-Time Operating System
SymTA/S:	Symbolic Timing Analysis for Systems
WCET:	Worst-Case Execution Time
WCRT:	Worst-Case Response Time
XML:	Extensible Markup Language
XTC:	XML Timing Cookie