AFTER: A CASE tool to Assist in Fine-Tuning of Embedded Real-Time Systems

Gaurav Arora‡ and David B. Stewart‡

‡Satellite Networking Division, Hughes Network Systems, Germantown, MD 20876; Email: garora@hns.com

‡Dept. of Electrical Engineering and Institute for Advanced Computer Studies University of Maryland, College Park, MD 20742; Email: dstewart@eng.umd.edu

Abstract: AFTER (Assist in Fine-Tuning of Embedded Real-time systems) is an interactive analysis and predictor tool for embedded systems. It helps designers quickly identify timing problems and systematically fine-tune an application during and after the implementation phase of a product’s life cycle. The tool begins with raw timing data collected from an embedded system. It analyzes the data to provide a temporal image of the current implementation, highlighting actual and potential problems. The user then interacts with AFTER to obtain predictions on what overall effect can be expected if small adjustments are made to configuration parameters or to the timing properties of software components. The tool integrates and extends prior research in scheduling, task monitoring, and operating system design for real-time systems.

1. Introduction

AFTER (Assist in Fine-Tuning of Embedded Real-time systems) is an interactive analysis and predictor tool for embedded systems. It helps designers quickly identify timing problems and systematically fine-tune an embedded application during and after the implementation phase of a product’s life cycle.

The tool uses measured timing data from an automatic task profiling mechanism [9] to present a temporal image of the system to the engineer. After analyzing the data, it highlights both actual and potential problem areas. The engineer can then query AFTER with “what if” types of questions, such as what if I halve the frequency of task X?, or what if I used an aperiodic server instead of an interrupt handler for event Y?

AFTER then uses a combination of the real data collected from the embedded system with theoretical data generated by analytic models which incorporate the desired changes, to present predict a new temporal image. AFTER does not give definite answers; rather the new image tells the engineer whether a particular change is likely or not likely to solve the timing problem. Only likely solutions then need to be tried. This helps in reducing the number of iterations of code modification needed to fine-tune the application.

Various Computer-Aided Software Engineering(CASE) tools exist to help design real-time systems. AFTER is unique, however, in that it provides real-time scheduling analysis after implementation. In contrast, other tools provide real-time scheduling advice before implementation. The scope of AFTER relative to other similar tools is shown in Figure 1. CAISARTS [2] is a rule-based system to obtain expert assistance for all aspects of the design related to real-time scheduling. PERTS [7] is a prototyping environment for real-time systems. It contains schedulers and resource access protocols, together with a comprehensive set of tools for analysis, validation and evaluation of real-time systems.

The key contributions of this work are the following:

• Creation of a practical tool that assists in fine-tuning embedded real-time systems after implementation.

• Combining the theory of various research groups into a single practical tool, and extending the theory to use real data and analytical models that more closely represent embedded real-time systems.

The motivation for this work stems from our experience as part of a software development team for a satellite modem. In industry, cost is often the most important factor which directs the design decisions. A decision that must be made early during the design phase is selection of the target processor, so that the necessary hardware is available when implementation begins. The choice of processor is based on an estimate of the computing capabilities needed, and is usually selected as the lowest cost processor that meets the estimates.

Unfortunately, once implementation begins, it is too late to change the hardware, even if more accurate execution times reveal a permanently overloaded processor. In such cases, tedious debugging, testing, and trial-and-error fine-tuning is performed to identify the problems and find solutions. Often, this becomes the longest portion of the product life cycle before delivery. The goal of AFTER is to reduce the amount of time needed to debug the timing characteristics and fine-tune a system after has been implemented.

2. Description of the tool

AFTER is an evolving tool which incorporates a variety of theoretical and practical analysis to help a designer fine-tune an embedded system. As shown in Figure 2, the four major modules of AFTER are the following:

Figure 1: Target software life-cycle stages for various tools
Data Collection & Storage Unit (DCSU): AFTER operates on timing data obtained from a real working system. This unit directly interfaces with the embedded system and collects raw data from the system parameters are derived. The data can be collected from a monitoring tool, preferably built-in to the operating system such as Chimera’s automatic task profiling mechanism [9]. If not built-in, the user must first instrumentize the source code, then use an external monitoring tool to collect the data. Regardless of the monitoring technique, any type of raw data can be passed to AFTER by appropriately defining a filter module.

Filter Unit (FU): The filter units extract only the necessary values from the raw data supplied by the DCSU, such as execution time, period of tasks, minimum inter-arrival time of interrupts, and operating system overhead, and forwards the information to the analysis unit. The filter modules are monitor-specific, such that using a different monitoring mechanism results in only needing to create an appropriate filter module. The filter unit can also be changed to provide different kinds of information to the analysis unit, depending on the type of analysis to be performed. For example, one filter unit may only extract worst-case execution times, while another may extract the entire timing history of a particular task.

Analysis Unit (AU): The analysis unit is the core of AFTER. The unit is continually evolving, as analysis based on a variety of computational models are being incorporated. It operates in two modes. In the analysis mode, it uses only real data collected from the embedded system, and presents a temporal image to the developer. It performs a schedulability analysis, using a mathematical model that is consistent with the scheduling algorithm used in the underlying embedded system. Any actual problems, such as missed deadlines, are displayed to the user. In addition, potential problem areas, such as higher than expected utilization of the critical set, or abnormally long locking of the CPU due to an interrupt, are also reported. In the predictor mode, the AU uses a combination of real data and estimated data to predict how a particular fine-tuning operation might change the timing characteristics of the system. If the AU predicts that the change will likely result in an improvement in system performance, the developer can implement those changes, then reanalyze the system using real data. The developer does not need to implement any changes that are not likely to improve the system. Further details of this unit are described in the next section.

Parameter Modification Unit (PMU): The parameter modification unit allows the developer to interact with the AU, to request that the AU predict the outcome if one of a variety of fine-tuning optimizations are made. Examples of the types of modifications that can be made are the following: switching one or more interrupt handlers to aperiodic servers, or vice versa; changing the frequency of a task; lowering the execution time of a particular task by optimizing the code or

![Schematic Representation of AFTER](image)
removing some functionality; and using dynamic instead of static scheduling. A prototype PMU graphical user interface has been implemented using Tcl/Tk.

3. Details of the Analysis Unit

The core of AFTER is a timing analyzer which uses established real-time systems theory to analyze an embedded system. It first uses only real data to analyze the current state of the system. Then, after receiving input from the PMU, it makes predictions about system performance in response to the parameter changes proposed by the developer.

Katcher et. al. [4] made scheduling theory more practical by taking into account operating system overhead. However, their theoretical contributions were not verified on real applications. The AU is an implementation based on Katcher’s computations, with some extensions in order to perform accurate analysis and to make meaningful predictions [1].

For testing our first prototype, we instrumentized a commercial satellite modem embedded application that was not behaving as expected during its alpha testing phase. The application uses fixed priority periodic tasks and interrupts. We used AFTER to determine whether using dynamic priority scheduling or aperiodic servers would improve the application, and to identify which software modules needed to be optimized most.

The prototype system runs on a VRTX32 software platform. Based on Katcher’s conventions, we derived the VRTX32 analytical model, which can be categorized as non-integrated event-driven scheduling. It is similar to non-integrated interrupt event-driven scheduling, except that VRTX32 task queue manipulations are initiated by system calls and not by external interrupts [10]. This results in a need to modify the way operating system overhead is measured and included into the mathematical model, as compared to Katcher’s analysis.

To create the VRTX32 model, we model an interrupt service routine (ISR) as a sporadic server [8] with its capacity being the maximum execution time of the ISR and its period equal to the minimum inter-arrival time of the interrupt. According to Sprunt [8], a sporadic server can be treated as a standard periodic task with the same period and execution time as the sporadic server. This allows us to add the effect of interrupts to standard fixed priority scheduling analysis equations. The schedulability test for a specific task set under VRTX32 is summarized by the following condition:

\[
S_i = \min_{j=n_{\text{intr}}+1} \left( \left( \sum_{i=1}^{n_{\text{intr}}} C_j + t \right) \right) \left( \frac{t}{T_j} \right) < 1
\]

where \( n_{\text{thr}} \) is the number of periodic threads, \( n_{\text{intr}} \) is the number of interrupts in the system, \( C \) and \( T \) are execution time and period of a periodic thread (or for an interrupt handler, \( T \) is the minimum inter-arrival time). The threads and interrupts are numbered in decreasing order of priority (with interrupt handler \( j=1 \) having highest priority in the system, and thread \( j=n_{\text{intr}}+1 \), having the highest priority among threads). \( t \) is the context switch overhead for a thread in VRTX32, and it includes the time to execute the scheduler and select the next thread. \( n_{\text{intr}} \) is the operating system overhead for servicing an interrupt.

3.1 Predictor Capabilities

The AU can operate as a predictor when some of the real data is replaced with hypothetical data. It allows a developer to modify certain application and system parameters, to obtain predictions about the likely affect that these changes will have on the system. The tunable options in our first prototype of AFTER are handling of aperiodic events as interrupts or aperiodic servers, choosing between static and dynamic scheduling, and modifying the period, frequency and execution time of a task.

Interrupts and Aperiodic servers: There is a trade-off between interrupt handlers and aperiodic servers. Interrupt handlers are typically non-preemptive and execute with the highest priority, which decreases the predictability of the system. An aperiodic server, on the other hand, can be used to improve predictability, but at the cost of higher operating system overhead.

Using the real data collected from the system, AFTER can predict the effect of converting one or more interrupt handlers to aperiodic servers, or vice-versa. To perform such predictions, we extend the model in equation (1). An aperiodic event is considered to be composed of two separate elements: a minimal interrupt handler which signals the arrival of the event, and an aperiodic server thread which performs the bulk of the computation in response to the event. The schedulability test for the prediction is the following:

\[
S_{\text{as}} = \max_{j=n_{\text{intr}}+1} \left( \sum_{i=1}^{n_{\text{intr}}+1} \left( C_j + \frac{t}{T_j} \right) \right) \left( \frac{t}{T_j} \right) < 1
\]

where \( K_{\text{sig}} \) is the overhead to signal an aperiodic server from the interrupt handler. AFTER is also capable of the reverse prediction. That is, given data from a system already using aperiodic servers, determine the schedulability of the system if some of the servers are converted to interrupt handlers.

Static and Dynamic Scheduling: This option of the AU can assist a developer in evaluating the benefit of switching from a static to a dynamic scheduler, and vice-versa. If a system has interrupts or possible transient overloads, there is no guarantee that using a dynamic scheduler will improve the system. Before making such a change in the system, the developer can observe the likely benefits by using AFTER, and only implement the switch in scheduling strategies if there are worthwhile benefits. If AFTER suggests that such a change is not worthwhile, then the developer can save a complete iteration.
of unnecessarily modifying the code, which could easily have taken several hours or days to implement.

To predict the effect of switching scheduling algorithms, we derived an analytical model for a system consisting of fixed priority interrupt handlers and dynamic priority tasks. The following equation predicts the outcome of scheduling the task set in a dynamic priority system, given the raw data collected from an embedded application that used fixed priority scheduling:

\[ n_{thr} = \sum_{i=1}^{n} \left( \frac{t}{T_i} \right) \left( C_i + \tau_{\text{task}} \right) + \sum_{j=1}^{n_{\text{intr}}} \left( \frac{t}{T_j} \right) \left( C_j + \tau_{\text{intr}} \right) \]  \( t \)  \( (3) \)

where \( n \) is the number of periodic threads and \( k \) is the number of high-priority interrupt handlers and \( \tau_{\text{task}} \) and \( \tau_{\text{intr}} \) are the operating system overhead for context switches and servicing an interrupt respectively in a dynamic priority system. Equation (3) is derived from the mixed priority case of Liu & Layland [6], and includes operating system overhead as per Katcher’s method [4]. We applied Jeffay & Stone’s [3] analysis for incorporating interrupt handling overhead, to obtain a non-recursive computation for equation (3), as opposed to Katcher’s recursive computations.

**Frequency and Period:** AFTER allows a developer to request a prediction on whether or not a system will be schedulable if the period or frequency of one or more tasks is modified. In some cases, changing the frequency of tasks can improve the system performance, while in other cases (such as when using the rate monotonic algorithm), reducing the frequency of some tasks could in fact lower the schedulable bound [5].

**Execution Time:** AFTER can predict the effect of changing the execution time of a task. If after trying all possible modifications to the system parameters, the developer finds that the CPU is still overloaded, the remaining option is to reduce the execution time of one or more tasks, either through optimization, by making some tasks soft real-time, or by removing some non-essential functionality. In either case, a major development effort is required to perform such modifications. Unfortunately, even after this lengthy process, there is no guarantee that the system will meet the timing requirements.

A major problem is that the designer does not have specific goals as to which modules need to be optimized, and more importantly, by how much, in order to make a difference in the overall schedulability of the system. AFTER can be used by the designer to estimate how much optimization is needed. For example, AFTER can help the developer determine that “if you reduce the execution of task A by 1 msec, and task B by 0.5 msec, then the system is likely to work; optimizing task C does not make any difference.” This type of information is valuable to the designer, as there is now a clear goal for the optimization, and saves the designer from needlessly spending time in areas that do not have any effect on the overall system timing.

4. Summary

AFTER is an analysis and predictor tool which helps designers quickly identify and fix timing problems in an embedded system after it has been implemented. During the latter stages of a software product cycle, developers may be faced with the task of fine-tuning a system which is not meeting all of its timing requirements. In some cases, the only thing needed would be a simple change of a task’s period, but developers are reluctant to do that because it is quite difficult to gauge the effect of that change on other tasks in the system. In other cases, the modifications are more drastic and require significant effort in optimizing some code, but the designer does not know which threads must be optimized, nor how much optimization is needed.

To address these issues, AFTER uses raw timing data collected from the embedded system, analyzes it, then provides a temporal image of the current implementation, highlighting actual and potential problems. AFTER is then used interactively through a graphical use interface to help the developer fine-tune the application effectively.

5. References


