Hierarchical Specification Methods for Platform-Based Design

Kai Richter, Rolf Ernst
Institute of Computer Engineering
Technical University of Braunschweig
D-38106 Braunschweig, Germany
{richter|ernst}@ida.ing.tu-bs.de

Wayne Wolf
Department of Electrical Engineering
Princeton University
Princton, NJ 08544, USA
wolf@ee.princeton.edu

Abstract

This paper presents a new methodology for the hierarchical design of embedded systems including jitter for quality-of-service (QoS). Hierarchical top-down design requires that the specifications for the systems must be translated into specifications for the components such that, when the components are implemented to meet their specifications, the system will also meet its specifications. In a platform-based design, the properties of some components will be known at the start of design while the properties of other components will not be known until after they are implemented. We can, however, use our knowledge about existing components to translate the system specification into specifications for the new components that must be implemented. This paper provides a methodology for top-down translation of system specifications into component specifications.

1 Introduction

Embedded computing systems exhibit complex behaviors and implement those behaviors using complex architectures. Writing correct specifications is a critical early step in system design. Design methodologies typically concentrate on the system specification, which is derived from user input, etc. However, the system specification is not sufficient to build the system. We must also specify the components of the system. If the components are miss-specified, then they will not work together to implement the system when they are connected together. This paper uses a design example to illustrate a new methodology for the derivation of component specifications from a system specification.

We are concerned with the non-functional aspects of the specification, namely the performance of the system and the components. Performance specifications are particularly challenging to consider in system design, since we cannot easily verify them by simulation. Our analysis method provides conservative bounds on the execution times required for components such that, if the component implementations satisfy their specifications, the complete system will also satisfy its overall specification. In contrast to many other performance analysis approaches like [5], we impose upper *and* lower bounds on response times of tasks in order to meet jitter constraints.

This methodology is particularly useful in platform-based design. A platform defines some parts of an implementation while leaving others up to the designer. The system designer therefore has two tasks when using a platform: ensuring that the system specification can be met by the platform; and deriving the specifications for the components not given in the platform. The specifications for the components to be added to the platform during design must be consistent with the other component specifications and the overall system specification. Our methodology ensures that all the component and system specifications will be consistent. This is also important for IP integration. Here, knowledge about the timing of an IP component needs to be provided to the designer. On the other hand, the knowledge about constraints narrows the IP component selection options in advance and reduces the number of design and analysis cycles.

The next section surveys related work. Section 3 contains the basic definitions which are used throughout the paper. The following section illustrates our methodology with a design example. We conclude the paper with an outlook on future work.

2 Related Work

Blazewicz [2] derives hierarchical deadlines for earliest-deadline first scheduling. RATAN [1] is a methodology for the rate analysis and time budgeting of periodic systems. We know of no other research that has developed derivations for hierarchical specifications that takes into accounts multi-rate systems, preemption, and jitter. Rate Monotonic Scheduling (RMS) [5] is a well-known real-time scheduling algorithm. Our model uses a somewhat more general model: we do not require the deadline to be at the end of the period and we allow jitter in arrival times. Lehoczky [4] provides some useful formulas for the analysis of more general static priority systems.

The importance of platform-based design has been extensively investigated in [3], and the software architecture has been identified as an integral part of the platform. More recent work like [6] concentrates on safe integration of IP components by providing timing data, and designers can choose among several implementations with different timing. However, there is, to the best of our knowledge, no concept of deriving detailed timing properties top-down.

3 System Specifications

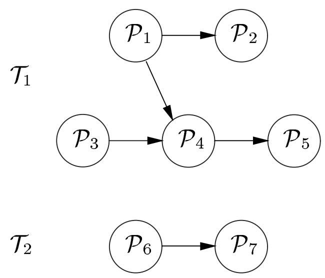

Figure 1: A task graph.

For the analysis throughout this paper, a system is specified by a **task graph** as depicted in Figure 1. Each node in the task graph represents a **process** $\mathcal{P}_i$. Edges in the task graph represent data dependencies. Each process $\mathcal{P}_i$ has a worst-case (maximum) execution time $C^+_{\mathcal{P}_i}$. We refer to the best-case (minimum) execution time of process $\mathcal{P}_i$ as $C^-_{\mathcal{P}_i}$.

Processes related by data dependencies form a **task** in the task graph. The tasks execute periodically. Every task $\mathcal{T}_i$ has a period $T_{\mathcal{T}_i}$. All the processes in a task have the same period. We assume no relationship between the periods. Each task $\mathcal{T}_i$ also has a deadline $D_{\mathcal{T}_i}$. Input and output actions happen in **event sequences**. A single element of a sequence is called an **event** e_j. We use the variable $t(e_j)$ to describe the time of e_j. An event occurs in the range $< t^-(e_j), t^+(e_j) >$. Successive events occur periodically, with the **period** between two elements (k and $k+1$) of the event stream e_j being

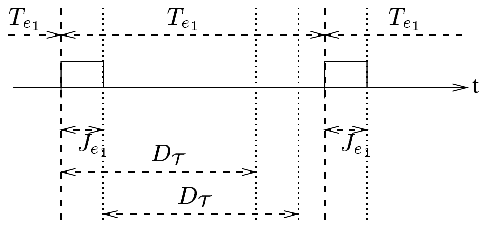

Figure 2: Events and Jitter.

$$\begin{aligned} T_{e_j} &= t^+(e_j(k+1)) - t^+(e_j(k)) \\ &= t^-(e_j(k+1)) - t^-(e_j(k)). \end{aligned} \quad (1)$$

The **jitter** of an event is the distance between the earliest and latest possible times of the event: $J_{e_j} = t^+(e_j(k)) - t^-(e_j(k))$.

The deadline for a task or process is measured from the actual arrival time of an input event $e_{\text{in}}(k)$ to the arrival time of the resulting output event $e_{\text{out}}(k)$:

$$t(e_{\text{out}}(k)) \leq t(e_{\text{in}}(k)) + D_\mathcal{T} \quad (2)$$

We assume that a task completes before it is reinitiated, so task execution is not pipelined. This means that

$$D_\mathcal{T} <= T_{e_1} - J_{e_1}, \quad (3)$$

where e_1 is $\mathcal{T}$'s input event. This is shown in Figure 2.

We can also define the jitter of a process $\mathcal{P}$ as the difference between the worst-case and best-case execution times: $J_\mathcal{P} = C^+_\mathcal{P} - C^-_\mathcal{P}$.

The **specification** for a system should describe its input/output characteristics. The specification should not unnecessarily constrain the implementation. We define the specification of a system as having three major elements:

- The task graph, including processes and data dependencies.
- The periods of the input and output events.
- The maximum jitters on the input and output events.

The system is specified in terms of the events, not the processes. The processes must satisfy the I/O requirements of the events. We use our methodology to translate event-oriented specifications into process-oriented specifications that we can use to drive process implementation.

The system specification includes both **properties** and **constraints**. A property describes an input—the periods and jitters of the input events, for example, are given properties of the environment. A constraint describes a requirement on an output—the periods and jitters of the output events are determined by the operation of the system. The implementation must obey the properties and operate so as to satisfy the constraints.

As a result, the system specification can be split into behavior, properties, and constraints:

- **behavior** Task graphs.
- **properties** Periods and jitters for all the input events.
- **constraints** Required periods and jitters for all the output events.

We may also have properties describing the implementation of some of the system components. That information is in the form of the worst-case and best-case execution times for processes. Before we can implement the system, we must have specifications for all its components. Those specifications must be consistent with the system specification. This requires that we must derive the properties for all the components—the process' worst-case and best-case execution times. In order to derive these execution times, we will in general have to derive the properties of input events and constraints on output events internal to the system—the inputs to and outputs of the component processes.

Our hierarchical design problem is to take the system behavior, properties, and constraints, add in the known component properties, and derive the required properties and constraints on the remaining components.

4 Example

We introduce our methodology with an example. Figure 3 shows a sample task graph with five processes grouped into two tasks $\mathcal{T}_1$ and $\mathcal{T}_2$.

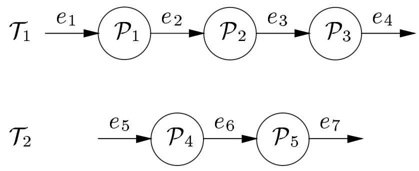

Figure 3: An example task graph including the events.

We assume in this section that processes are scheduled using static priorities that do not change during execution. Processes in a task have the same priority but each task has a unique priority. We denote the priority of a task $\mathcal{T}$ as $P_\mathcal{T}$ and, for convenience, the priority of process $\mathcal{P}$ as $P_\mathcal{P}$. We ignore context-switching overhead.

In this simple example, we are given the specification for the system:

- The two task graphs $\mathcal{T}_1$ and $\mathcal{T}_2$.
- The periods and jitters of the input events e_1 and e_5.
- The periods and jitters of the output events e_4 and e_7.

We are also given information on the implementation of all but one component:

- The best-case and worst-case execution times of $\mathcal{P}_1$, $\mathcal{P}_3$, $\mathcal{P}_4$, and $\mathcal{P}_5$.

We want to determine properties and constraints for $\mathcal{P}_2$:

- The best-case and worst-case execution times for $\mathcal{P}_2$.

4.1 Case 1: Single Task

In order to formulate some basic formulas before going on to the more general case, let us assume that $P_{\mathcal{T}_1} > P_{\mathcal{T}_2}$. This means that $\mathcal{T}_2$ cannot preempt $\mathcal{T}_1$, simplifying the construction of the relations. We will see in the next subsection how preemption affects the construction of the relations that describe the system.

Since $\mathcal{T}_2$ does not interfere with $\mathcal{T}_1$, the formulas that describe $\mathcal{T}_1$ are relatively straightforward. Because task execution is not pipelined, we can derive the information we need about the specification for $\mathcal{P}_2$ by determining the jitters of its input event e_2 and its output event e_3.

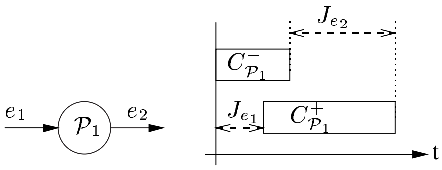

Figure 4: Propagation of jitter without preemption.

We can derive the jitters for e_2 and e_3 from the system's jitter specifications and the specifications of the known components $\mathcal{P}_1$ and $\mathcal{P}_3$. As depicted in Figure 4, we determine $\mathcal{P}_2$'s input jitter forward from the system inputs:

$$\begin{aligned} J_{e_2} &= J_{e_1} + C^+_{\mathcal{P}_1} - C^-_{\mathcal{P}_1} \\ &= J_{e_1} + J_{\mathcal{P}_1} \end{aligned} \tag{4}$$

Analogously, we derive $\mathcal{P}_2$'s output jitter requirement backwards from the system outputs:

$$J_{e_3} = J_{e_4} - J_{\mathcal{P}_3}. \tag{5}$$

In this case, we subtract $J_{\mathcal{P}_3}$ since jitter monotonically decreases from output to input.

Given the jitters for $\mathcal{P}_2$'s input and output events, we can determine the maximum allowable jitter for the process itself:

$$J_{\mathcal{P}_2} = J_{e_3} - J_{e_2}. \tag{6}$$

Given the process jitter, the task deadline, and the execution times for the other processes, we can determine the allowable best-case and worst-case execution times for $\mathcal{P}_2$:

$$C^+_{\mathcal{P}_2} \leq T_{\mathcal{T}_1} - J_{e_1} - C^+_{\mathcal{P}_1} - C^+_{\mathcal{P}_3} \tag{7}$$

$$C^-_{\mathcal{P}_2} \geq C^+_{\mathcal{P}_2} - (J_{e_3} - J_{e_2}) \tag{8}$$

Note that $C^+_{\mathcal{P}_2}$ is the constraining upper bound on the execution time of $\mathcal{P}_2$ to fulfill performance requirements from Equation 3. The minimum allowable execution time is determined relative to the actual value of $C^+_{\mathcal{P}_2}$ by the need to deliver $\mathcal{P}_2$'s output no earlier than the beginning of its output jitter window. If the process actually needs less time, it must wait to deliver its output. This may be necessary, for example, to avoid overflowing queues.

4.2 Case 2: Interacting Tasks

We now consider a more complex case: $P_{\mathcal{T}_2} > P_{\mathcal{T}_1}$. In this case, the execution times of the processes in $\mathcal{T}_2$ must be introduced into the formulas that describe $\mathcal{T}_1$.

If $\mathcal{T}_2$ has higher priority, then it can preempt $\mathcal{T}_1$'s execution. As a result, the jitter of $\mathcal{T}_1$ no longer depends solely on the properties of $\mathcal{T}_1$'s events and processes—it also depends on the worst-case interference of $\mathcal{T}_2$.

Once again, we start by deriving the jitters for e_2 and e_3. However, in this case, the jitters must reflect the delays introduced by possible preemptions by $\mathcal{T}_2$. The jitters for $\mathcal{P}_1$ and $\mathcal{P}_3$ must include these preemption effects, which are described by $J_{\mathcal{P}_1,\mathcal{T}_2}$ and $J_{\mathcal{P}_3,\mathcal{T}_2}$. However, defining these preemption-induced jitters is not trivial. In order to define the effects of preemption on $\mathcal{P}_1$ and $\mathcal{P}_3$, we must first describe the effects of preemption on the entire task $\mathcal{T}_1$. We do so by defining t_{123} as the accumulated execution time of $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ including worst-case preemption:

$$t_{123} = C_1 + C_2 + C_3 + J_{\mathcal{T}_1,\mathcal{T}_2}. \tag{9}$$

Note, that this general equation does not reflect execution time intervals. Upper and lower bounds on the parameters are later used to obtain the actual values for $C^+_{\mathcal{P}_2}$ and $C^-_{\mathcal{P}_2}$. We can now use an iterative equation (similar to that used by Lehoczky et al. [4]) to find $J_{\mathcal{T}_1,\mathcal{T}_2}$:

$$J_{\mathcal{T}_1,\mathcal{T}_2} = \frac{t_{123}}{T_{\mathcal{T}_2}} C_{\mathcal{T}_2}. \tag{10}$$

This form of equation is well-known in Rate Monotonic Analysis to analyze CPU response time. However, we are not interested in task response time or even directly in the jitters on the events. In order to create a specification for $\mathcal{P}_2$, we need to find $C^+_{\mathcal{P}_2}$ and $C^-_{\mathcal{P}_2}$.

We start by finding $C^+_{\mathcal{P}_2}$. Since we are looking for maxima, we use upper execution time bounds in equations 9 and 10 to obtain the worst-case joint execution time for $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ in this form:

$$t^+_{123} = C^+_{\mathcal{P}_1} + C^+_{\mathcal{P}_2} + C^+_{\mathcal{P}_3} + n^+_{\mathcal{T}_2} C^+_{\mathcal{T}_2}. \tag{11}$$

$n^+_{\mathcal{T}_2}$ is the number of times that $\mathcal{T}_2$ preempts $\mathcal{T}_1$:

$$n^+_{\mathcal{T}_2} = \left\lceil \frac{t^+_{123} + J_{e_5}}{T_{\mathcal{T}_2}} \right\rceil \tag{12}$$

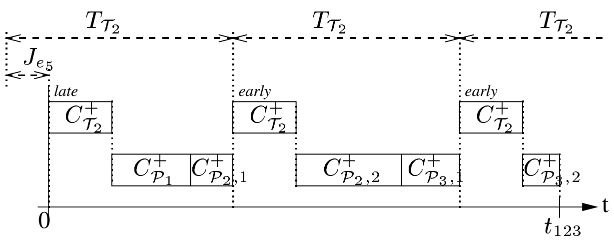

Figure 5: Maximum interference of task $\mathcal{T}_2$.

Since t^+_{123} is the total time over which $\mathcal{T}_2$ can preempt, we can determine the number of preemptions by dividing that time interval by $\mathcal{T}_2$'s period. A similar analysis is performed in Rate Monotonic Analysis. We add J_{e_5} to t^+_{123} to capture the worst-case influence of the jitter, since the worst-case interference by task $\mathcal{T}_2$ occurs when the first task execution is late and all others are early, as shown in Figure 5.

We want to find the maximum allowable $C^+_{\mathcal{P}_2}$ in order to give the designer the most time to implement the process' function. We are interested in a conservative value for $C^+_{\mathcal{P}_2}$, such that the system will always work properly; larger values could result in the system working in some cases but not others.

In order to find the upper bound $C^+_{\mathcal{P}_2}$, we define t^+_{123}, which is the latest time for which $\mathcal{P}_2$'s specification satisfies the system's requirements. Clearly,

$$C^+_{\mathcal{P}_2} \leq t^+_{123} - C^+_{\mathcal{P}_1} - C^+_{\mathcal{P}_3} - n^+_{\mathcal{T}_2} C^+_{\mathcal{T}_2}. \qquad (13)$$

We need to find t^+_{123}, from which we can find $n^+_{\mathcal{T}_2}$, in order to find $C^+_{\mathcal{P}_2}$. That maximum time is derived from $\mathcal{T}_1$'s deadline. We know from the above definitions that

$$t^+_{123} \leq D_{\mathcal{T}_1}. \qquad (14)$$

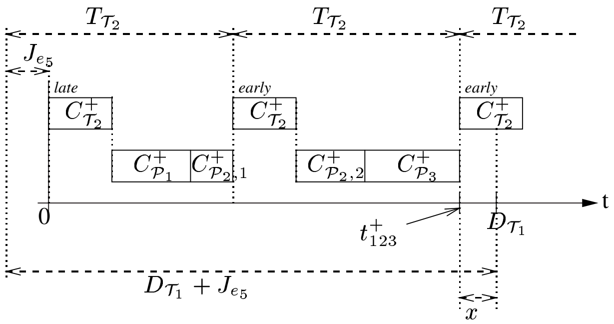

Figure 6: Determining t^+_{123}.

However, we need an exact value for t^+_{123}. There are two cases to consider: either $\mathcal{T}_2$ is executing at the deadline or it is not. If $\mathcal{T}_2$ is executing at the deadline of $\mathcal{T}_1$, then t^+_{123} is at the beginning of that last execution of $\mathcal{T}_2$ before the deadline, as shown in Figure 6; since $\mathcal{T}_1$ cannot execute at all while $\mathcal{T}_2$ executes, then the $\mathcal{T}_1$ only has the earlier interval to execute. If $\mathcal{T}_2$ is not executing, then t^+_{123} is in fact equal to the deadline since it receives no interference at that point. We must also include the jitter of $\mathcal{T}_2$. We can describe these situations mathematically as follows. Let

$$x = (D_{\mathcal{T}_1} + J_{e_5}) \bmod T_{\mathcal{T}_2}. \qquad (15)$$

Then

$$t^+_{123} = \begin{cases} D_{\mathcal{T}_1} - x & x < C^+_{\mathcal{T}_2} \\ D_{\mathcal{T}_1} & \text{otherwise} \end{cases} \qquad (16)$$

Therefore, we can find the upper bound on $C^+_{\mathcal{P}_2}$ as

$$C^+_{\mathcal{P}_2} \leq t^+_{123} - C^+_{\mathcal{P}_1} - C^+_{\mathcal{P}_3} - \left\lceil \frac{t^+_{123} + J_{e_2}}{T_{\mathcal{T}_2}} \right\rceil C^+_{\mathcal{T}_2}. \qquad (17)$$

Notice that, unlike the case in Rate Monotonic Analysis, we can find a closed-form solution for our equation. This is because we have imposed the additional constraint of requiring the maximum possible execution time for the process—we are looking for a specification on the response time, not the actual response time. In contrast, the RATAN [1] methodology uses the opposite approach. Time-budgets for tasks are derived first. Subsequently, an analysis checks whether they are fulfilled by the implementation. Furthermore, RATAN does not provide specifications of processes (sub-tasks).

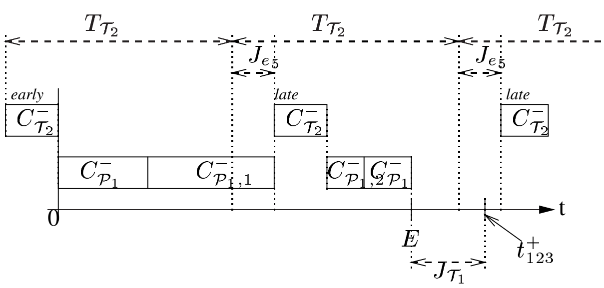

Figure 7: Minimum interference of task $\mathcal{T}_2$.

We must also find the smallest possible minimum execution time in order to minimize $\mathcal{P}_2$'s internal buffering. Consider the analysis shown in Figure 7. The reasoning is similar as for the maximum case, but since we are dealing with minima, we are interested in the minimum number of starts of $\mathcal{T}_2$ and we must consider minima, not maxima. The minimum computation time occurs when the first execution of $\mathcal{T}_2$ is early and all the other task executions are late.

$\mathcal{T}_2$ can preempt $\mathcal{T}_1$ this many times:

$$n^-_{\mathcal{T}_2} = \left\lfloor \frac{t^-_{123} - J_{e_5} + C^-_{\mathcal{T}_2}}{T_{\mathcal{T}_2}} \right\rfloor \tag{18}$$

We must include $C^-_{\mathcal{T}_2}$ because, as can be seen in Figure 7, we must complete an execution of $\mathcal{T}_2$ since it has higher priority. We find t^-_{123} relative to t^+_{123}. We need a correction term similar to the max case to consider whether $\mathcal{T}_2$ preempts. Rather than work from the deadline, as in the max case, we must work from the maximum available execution time, excluding the jitter:

$$E = t^+_{123} - (J_{e_4} - J_{e_1}). \tag{19}$$

Let $y = (E - J_{e_5}) \bmod T_{\mathcal{T}_2}$ and $z = T_{\mathcal{T}_2} - y$. Then

$$t^-_{123} = \begin{cases} E + z & z \leq C^-_{\mathcal{T}_2} \\ E & \text{otherwise} \end{cases} \tag{20}$$

Therefore, we can find the lower bound on $C^-_{\mathcal{P}_2}$ as

$$C^-_{\mathcal{P}_2} \geq E - C^-_{\mathcal{P}_1} - C^-_{\mathcal{P}_3} - \left\lfloor \frac{t^-_{123} - J_{e_5} + C^-_{\mathcal{T}_2}}{T_{\mathcal{T}_2}} \right\rfloor C^-_{\mathcal{T}_2}. \tag{21}$$

5 Conclusions

We have used an example to illustrate our methodology for top-down system specification. Our methodology allows us to derive specifications for components of the system that are consistent with the specification for the system. The example shows that deriving component specifications is non-trivial when the system allows preemption. But because the results of the analysis are closed-form, they provide useful information to the component designer.

Previous work concentrated on answering questions about an implementation; in contrast, our work provides the missing link between system specification and component specification in top-down design. The current model is sufficient to sufficiently general to show that, even before implementation begins, deriving useful specifications for the components of the system is a non-trivial task. We plan to extend our process model and to generalize our scheduling assumptions.

References

[1] R. K. Gupta A. Mathur, A. Dasdan. Rate Analysis for Embedded Systems. *ACM Transactions on Design Automation of Electronic Systems*, 3(3):408 – 436, July 1998.

[2] J. Blazewicz. *Modeling and Performance Evaluation of Computer Systems*, chapter Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines. North-Holland, Amsterdam, 1976.

[3] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd. *Surviving the SOC Revolution*. Kluwer Academic Publishers, 1999.

[4] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings Real-Time Systems Symposiom*, pages 166–171, IEEE Computer Society Press, 1989.

[5] 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.

[6] T. Zhang, L. Benini, and G. De Micheli. Component selection and matching for IP-based design. In *Proc. Design, Automation and Test in Europe - DATE*, pages 40–46, 2001.