A General Approach to Modeling System-Level Timing Constraints

Marek Jersak, Dirk Ziegenbein, Rolf Ernst
Technische Universität Braunschweig
jersak|ziegenbein|ernst@ida.ing.tu-bs.de

Abstract

Timing constraints are an integral part of the design of embedded real-time systems. Initially, timing constraints are imposed by the system environment and performance requirements. They affect many system-level implementation decisions, in particular hardware/software partitioning, allocation and scheduling, which in turn may lead to additional constraints for other system parts. Different system-level analysis and implementation techniques assume different types of constraints as well as specification models and target architectures. Such 'proprietary' solutions inhibit the combination of different models and techniques for system-level design, which is increasingly important due to the growing embedded system complexity and heterogeneity. In this paper, we propose an efficient representation for timing constraints that is flexible enough to model constraints required by modern analysis and implementation techniques, in particular in the presence of event jitter and burst.

1 Introduction

For the modeling, analysis and implementation of complex embedded real-time systems, it is becoming increasingly important to combine different modeling languages, methodologies and tools (e. g. [3]) to be able to cope with growing system complexity and heterogeneity. However, different 'proprietary' assumptions made by modern system-level analysis, hardware-software partitioning, allocation and scheduling techniques about specification models, target architectures and constraints currently inhibit this combination. In particular the modeling of complex timing constraints such as maximum jitter or maximum burst length is modeled in a proprietary way. In the following the main requirements for generally applicable modeling of timing constraints are described.

Natural Level of Abstraction and Ease of Use Modeling of timing constraints is desirable at a natural level of abstraction. It has to go beyond the traditionally specified execution rate constraints or individual process deadlines and additionally allow to model original constraints like input-output latency constraints or realistic implementation-dependent constraints, such as maximum sampling time jitter or minimum signal distances during a burst. Furthermore, constraint modeling should be easy for a designer and machine readable. It should also be possible to extend the types of constraints that can be specified to allow adaption to emerging analysis techniques.

Language- and Technique-Independence System-level languages and models of computation abstract timing issues of an implementation. Either only partial (e.g. in data flow models) or complete (e.g. in C) ordering of operations is considered, or time is idealized into a fixed-step scheme and zero delay execution as is the case in synchronous languages or in periodic simulation models (e.g. Matlab/Simulink). Typically, timing constraints are not part of these languages but rather are provided in an additional natural language document. They need to be formalized early in the design flow to be applicable to system-level design techniques. At the same time, language-independence should be maintained to allow e.g. latency path constraints across language boundaries. Over-constrained timing as in the case of Simulink must be relaxed, and replaced by the designer with timing constraints that really matter [7].

It should be possible to specify timing constraints in a design technique-independent way in order to support a heterogeneous design flow where different design techniques can even be provided by different sources. This would allow evaluating several implementation solutions, and potentially selecting different ones for different system parts. Particular emphasis has to be placed on techniques that consider jitter and bursts [6, 9] which are introduced e.g. by bus contention.

Source Classification Timing constraints can be divided into constraints imposed by the environment in which an embedded system is supposed to operate and constraints that are a result of design decisions. The former include I/O latency constraints, maximum sampling jitter or minimum execution rate constraints. Design decisions, in particular scheduling and allocation, constrain the timing for the remaining system. Similarly, in a 'divide-and-conquer' approach new timing constraints have to be specified for each subsystem. Consequently, those timing constraints that may be altered have to be distinguished from those that may not be altered in order to support an iterative design flow.

The remainder of the paper is organized as follows. After an overview of related work we present our timing constraint modeling in Section 3, where we also show how to combine it with a system representation suitable for system-level analysis. Our approach is further demonstrated in Section 4 by targeting a particular analysis technique and by showing the potential for analysis based on a suitable modeling of occurring bursts.

2 Related Work

There is a large number of system-level analysis and scheduling techniques for embedded real-time systems. Traditional scheduling techniques, e.g. Rate-Monotonic Scheduling, are restricted to periodically arriving events, periodic deadlines and periodic rates. They usually assume independent processes (or tasks) and allow analysis based on Worst-Case Execution Times (WCET). For an extensive overview of existing scheduling techniques and schedulability tests see Fidge [4]. Ernst [2] provides a similar overview for techniques used for hardware/software codesign. As has been shown e.g. by Gerber [5] or Yen [10], best-case values are needed to analyze real-time scheduling anomalies in distributed systems. This requires using *intervals* where values are delimited by an upper and lower bound. Other work, e.g. by Tindell [9] or Gresser [6], acknowledges the fact that realistic systems exhibit communication jitter and bursts. Our goal is to specify timing constraints such that different techniques can be applied, extended and combined as appropriate.

Many mathematical formalisms have been developed to model systems with time, including timed process algebras, formalisms based on temporal logic and timed constraint programming languages. A brief overview is given in the introduction to [1]. All of them offer ways to reason about *functional correctness* in the presence of time to avoid *value failures*. However, real-time issues of an actual implementation are not considered. For example, the timed concurrent constraint language, *tccp* [1], is a formalism consisting of operational semantics (i.e. a transition system) and a denotational model. Actions are governed by the presence (absence) of certain global constraints, and in turn can alter the constraints 'store'. This allows to specify 'what should happen when', e.g. delayed execution, time-outs or watchdogs. However, it would be very complicated to model typical implementation decisions affecting timing, e.g. dynamic scheduling.

Emerging system languages such as SystemC or SpecC allow to describe or assert process and communication timing *behavior* for the purpose of simulation or IP exchange. Timing information at this level is necessary since without it system-level timing constraints cannot be validated, but the constructs themselves are not suitable for *modeling* system-level timing constraints. Recently, Rosetta [8] has been introduced as a requirement specification language. Rosetta offers a multi-faceted view of a specification with sets of terms that an implementation is required to satisfy. One view is via the 'constraints' domain, which based on the little information currently available seems to support only simple latency constraints. This is not sufficient for system-level analysis.

Recently the SPI model [11, 12] has been developed which is well suited for system-level analysis in a heterogeneous design flow, since SPI captures in a homogeneous way those properties which are important for system-level analysis but abstracts functional details. In section 3 we will show ways to combine our constraint modeling with a system representation using the SPI model.

3 Timing Constraint Modeling

In this section we introduce our small set of parameterized timing constraint macros. By choosing parameter values appropriately, a large set of constraints with intervals, jitter and bursts can be specified. This is done using a textual notation in XML, since XML is is human- and machine- readable and supported by many existing XML tools. However, for illustration we use a graphical notation in this paper.

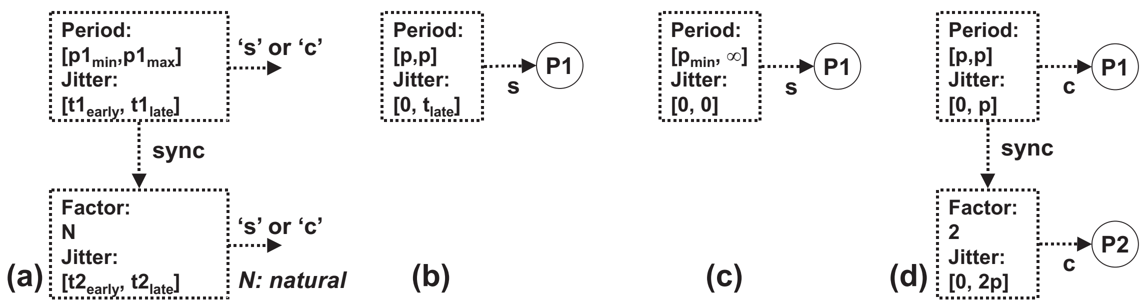

Figure 1. Rate Constraint macro (a) and some applications (b, c, d)

Rate Constraints are used to model constraints on the starting and completion time of processes. Fig. 1a shows the general macro. Depending on the choice of parameters, different types of constraints can be modeled. Fig. 1b shows a relaxed periodic start constraint (the period is fixed, but the starting time does not have to be exact). In Fig. 1c, a sporadic start constraint with a minimum distance is modeled. Fig. 1d shows two processes with a synchronized periodic completion constraint (deadline) where *P2*'s period is twice as large as *P1*'s. Explicit synchronization is necessary since no global clock is assumed. In XML notation, this constraint would be written as follows (dd="false" means that this constraint is *not* a result of a design decision and thus cannot be altered by undoing a previous implementation step).

```
<RC id="rc1" system="SPI.syst" dd="false">
    <Element name="P1" type="completion"/>
    <Period low="p" high="p"/>
    <Jitter low="0" high="p"/>
</RC>
```

```
<RC id="rc2" system="SPI.syst" dd="false">
    <Element name="P2" type="completion"/>
    <Sync master="rc1" factor="2"/>
    <Jitter low="0" high="2p"/>
</RC>
```

Latency Path Constraints are used to limit the time for causal process executions along a certain path. The macro is shown in Fig. 2a. *Burst Constraints* are used to model the activation of processes that display bursty behavior. Examples for such processes are a handler for packet arrivals or an abstract model for a communication network. We have adopted the terminology used by Tindell [9] and extended it with an explicit lower bound for intervals. The macro is shown in Fig. 2b.

Figure 2. Latency Path Constraint macro (a) and Burst Constraint macro (b)

3.1 Combination with the SPI model

The SPI (**S**ystem **P**roperty **I**ntervals) model [11, 12] is an internal representation that facilitates the safe integration of a heterogeneously specified system and enables system optimization across language boundaries. A major focus is to enable analysis of timing. In this section, only the basic concepts of SPI, which are necessary to understand this paper, are introduced informally.

Computational elements in SPI are processes that communicate via two different channel types, FIFO queues and registers. The activation of SPI processes is implicit and based on data availability, i. e. a process *may* start if there is sufficient data on its input queues to support one execution. The SPI model elements are not characterized by their exact behavior but by a set of parameters such as activation function, latency time, data rates (the number and size of data tokens consumed or produced on each channel per execution) and initial tokens on channels. Data is produced on and consumed from all connected channels within certain time intervals during the execution of a process. Parameters may be specified as intervals. For example, a data rate interval bounds the number of produced or consumed data, capturing data-dependent communication. Using the concept of process modes, conditional process behavior (which depends on internal states and input data) can also be modeled explicitly (see Sec. 4). *Virtual processes and channels* are used to model the system environment and to represent additional information, in particular scheduling dependencies and as we will see advanced timing constraints. Virtual elements do not have a direct functional correspondence but influence implementation.

Apart from a functional abstraction, also timing constraints can be specified in SPI but only in form of latency path constraints (LC). This was done to keep the model simple and general. It is possible to express other types of timing constraints using LCs and virtual elements, but this leads to complex constructs that can be difficult to understand and hard to analyze in particular when jitter and bursts are modeled. Expressing complex timing constraints separately and combining them with the SPI representation of a system is a much more efficient, modular and flexible approach. Because of the separation, our timing constraint macros can reference elements in any language and can be extended if necessary without affecting the language they reference. However, it can be advantageous to map our timing constraint macros to SPI LCs, since an analysis technique based only on LCs is automatically able to evaluate any other form of timing constraints, making this a very general approach. On the other hand, it is easier to apply existing analysis techniques using our macros if these techniques comprehend their semantic meaning directly or only with minor transformations. We will show this in the first example in next section.

As an example, the mapping of the rate constraint macro to SPI using SPI LCs and *virtual* elements is shown in Fig. 3. The LC on channel $Cs1$ forces a periodic execution of process $P1$ with the period bounded by $p1_{min}$ and $p1_{max}$. $P1$

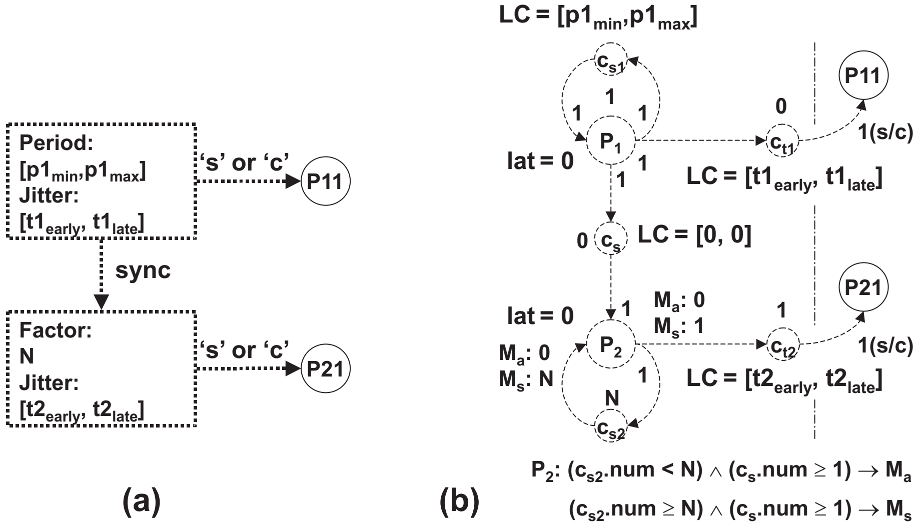

Figure 3. Rate constraint macro and its mapping to the SPI model

consumes one token from $Cs1$ and immediately (its latency is zero, a valid assumption for virtual elements) produces one token on channels $Cs1$, Cs, and $Ct1$. The token on $Ct1$ has to be consumed (either at starting or completion time of its execution) by process $P11$ within the LC interval on $Ct1$, producing the jitter allowed. The token on Cs is immediately consumed by process $P2$, which immediately produces one token on channel $Cs2$, and depending on its mode, may consume N tokens from $Cs2$ and produce on token on $Ct2$. Mode selection happens according to the function in Fig. 3b. This produces the desired synchronization.

4 Examples

Our first example shows how our timing constraint macros can be mapped to an existing analysis approach and how design decisions can affect timing constraints. The system in Fig. 4a models periodic processes with data dependencies, multiple deadlines and multiple release times using synchronized starting time constraints with jitter and latency path constraints. This representation is equivalent to the task graph model used in the response time analysis method for static priority scheduling by Yen and Wolf [10], which differs in assigning a single explicit deadline for the task graph and using virtual processes to model different release times and different deadlines for individual paths. The mapping is trivial.

To show how scheduling decisions affect timing constraints, note that the maximum execution time allowed for process $P1$ depends on the worst-case execution time of process $P3$, which can be longer than its core execution time because of interrupts by higher priority processes. Now assume that the highest priority has been assigned to process $P3$. Its execution time now equals its core execution time because it no longer can be interrupted. Thus, LC_1 can be replaced by a new latency constraint LC_1' = *[0, (dead*$_1$ *- core*$_{P3}$*)]* that is valid for the path $\{c1, P1, c4\}$.

In our second example (see Fig. 4b) we model a system containing an IP video phone decoder Pd that reads frames from a network (modeled as a bursty process Pn). The decoder is scheduled using a best-effort strategy, running on a single processor together with two periodic processes Pa and Pb with hard deadlines, i.e. frames may be discarded if the decoder is running out of time. We model the decoder using two modes. In mode Mf a frame is consumed from channel ci, decoded, and output on channel co. In mode Md a frame is consumed from channel ci and discarded. To avoid an overflow on channel ci we need to analyze if a run-time scheduler can always determine early enough when to discard frames. On the other hand, frames should not be unnecessarily discarded.

Upwards arrows in Fig. 5a show the worst-case timing of arriving frames (as early as possible). Downward boxes and arrows show worst-case analysis results assuming a channel ci that can buffer a maximum of two frames, i.e. how a scheduler would have to select between Mf or Md to avoid an overflow on channel ci based on worst-case response times of processes Pa, Pb and Pd. Since Pa and Pb have to complete execution every 15 time units, 3 time units are available per 15-time-unit period for Pd in the worst case before it has to be interrupted. Therefore, 5 periods are needed in the worst-case to execute Pd in mode Mf (interrupts are not shown in Fig. 5). Pd is executed at the beginning of each period to minimize response time. Fig. 5b shows the number of frames waiting in channel ci assuming that a frame is consumed at the end of execution of Pd.

Scheduling rules can be deduced from the analysis results. For example, after the arrival of the 3rd frame, the 2nd frame has to be discarded, since decoding might not be completed before the arrival of the 4th frame. The system can be analyzed accordingly with other buffer sizes ci. In a further analysis step, a statistical approach can be used to determine the number of frames that are dropped in a typical scenario based on the size of ci.

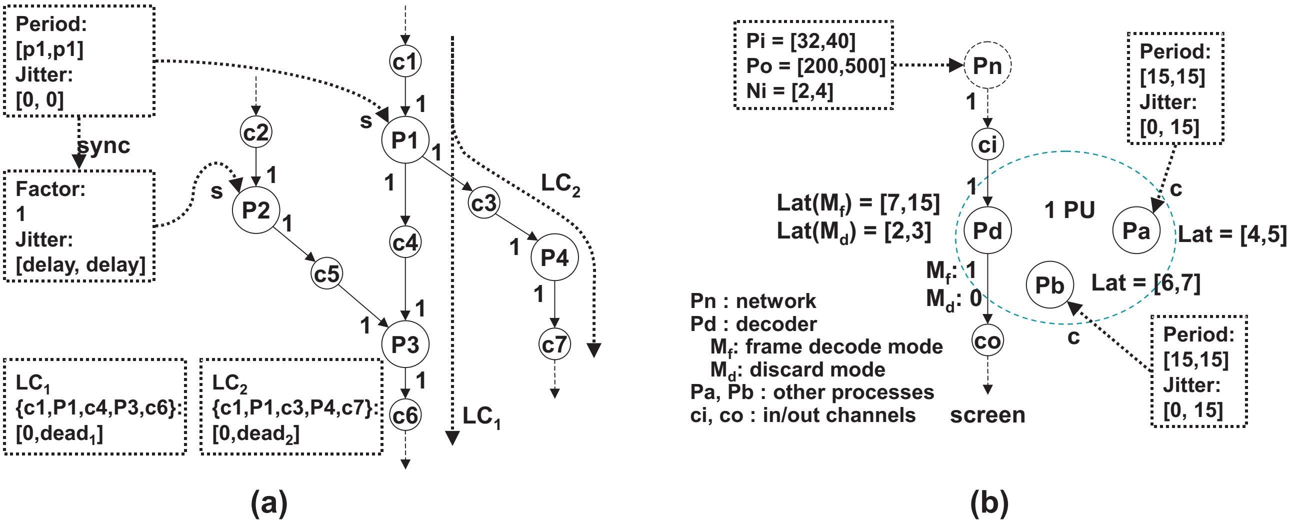

Figure 4. Periodic processes with data dependencies (a) and video phone decoder (b)

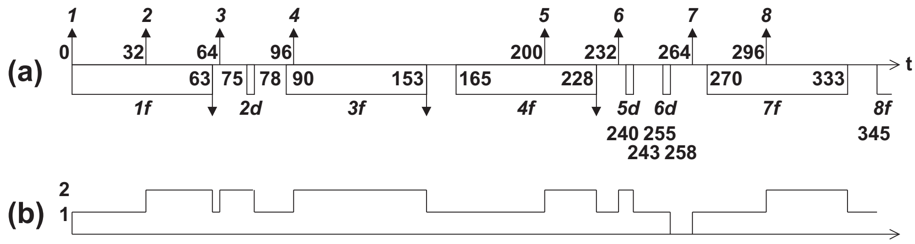

Figure 5. Video phone decoder: Worst-case timing with 2-frame buffer

5 Conclusion

In this paper we showed that a small set of timing constraint macros at a natural level of abstraction is sufficient to model a large variety of timing constraints required by modern system-level analysis and implementation techniques. The macros presented are language- and methodology-independent and thus facilitate the combination of multiple design languages and techniques, which is becoming essential because of rapidly growing embedded real-time system complexity. We demonstrated our approach by targeting the SPI system representation suitable for system-level analysis, a particular analysis technique and by showing the potential for analysis in the presence of jitter and bursts.

References

[1] F.S. de Boer, M. Gabbrielli, and M.C. Meo. A timed concurrent constraint language. *Information and Computation*, To appear.

[2] R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design & Test of Computers*, April 1998.

[3] R. Ernst and A. A. Jerraya. Embedded system design with multiple languages. In *Proc. 5th Asia South Pacific Design Automation Conference (ASP-DAC'00)*, January 2000.

[4] C. J. Fidge. Real-time schedulability tests for preemptive multitasking. *Real-Time Systems*, 14:61–93, 1998.

[5] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time taks. *IEEE Transaction on Computers*, 44(3):471–479, March 1995.

[6] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123, Oulu, Finland, 1993.

[7] Marek Jersak, Ying Cai, Dirk Ziegenbein, and Rolf Ernst. A transformational approach to constraint relaxation of a time-driven simulation model. In *Proceedings 13th International Symposium on System Synthesis*, Madrid, Spain, September 2000.

[8] Systems Level Design Language community. *Rosetta Reference*. http://www.sldl.org/.

[9] K. W. Tindell. An extendible approach for analysing fixed priority hard real-time tasks. *Journal of Real Time Systems*, 6(2):133–152, March 1994.

[10] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11), November 1998.

[11] D. Ziegenbein, R. Ernst, K. Richter, J. Teich, and L. Thiele. Combining multiple models of computation for scheduling and allocation. In *Proceedings Sixth International Workshop on Hardware/Software Co-Design (Codes/CASHE '98)*, pages 9–13, Seattle, USA, March 1998.

[12] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele. Representation of process mode correlation for scheduling. In *Proceedings International Conference on Computer-Aided Design (ICCAD '98)*, San Jose, USA, November 1998.