Integrated Modeling and Analysis of Computer-Based Embedded Control Systems

Zonghua Gu and Kang G. Shin
Real-Time Computing Laboratory
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109, USA
{zgu, kgshin}@eecs.umich.edu

Abstract

Embedded real-time systems are ubiquitous in modern society, many of which perform safety-critical functions, and therefore, it is imperative to have tools and techniques that can guarantee a high degree of system correctness. They typically perform information processing on a digital computer tightly coupled with the continuous physical environment. Even though hybrid systems is an active research area, most work has ignored the scheduling behavior of software processes due to contention for the shared CPU resource. We propose an integrated approach based on hybrid automata and model-checking for modeling and analysis of computer-based embedded control systems where real-time scheduling behavior of the controller software is explicitly represented at the model-level, together with the physical environment that it interacts with. An application example is used to demonstrate the benefits of the integrated approach in performing tradeoff analysis involving both the controller software and the controlled physical system.

1 Introduction

Embedded real-time systems are ubiquitous in modern society, many of which perform safety-critical functions, and therefore, it is imperative to have tools and techniques that can guarantee a high degree of system correctness. Real-time systems must satisfy both functional and non-functional properties, such as timing and resource constraints. In particular, addressing real-time issues is becoming the bottleneck in embedded systems development. Oftentimes it is not until the later phases of the development life-cycle when the developers discover that system timing properties are not satisfiable, requiring costly redesign and re-implementation, and resulting in schedule and budget overruns. It is important to use powerful real-time analysis tools and techniques at early design phases in order to avoid undesirable surprises at later testing and integration stages.

Embedded real-time systems typically perform information processing tightly coupled with physical processes. The boundary between physical and software processes are often blurred. However, modeling tools tend to focus on either one or the other. The Model-Integrated Computing [15] approach advocates integrated modeling:

> "Computers now control many critical systems in out lives...Such computers wed physical systems to software, tightly integrating the two and generating complex component interactions unknown in earlier systems. Thus, it is imperative that we construct software and its associated physical system so they can evolve together."

A computer-based embedded control system typically consists of one or more digital controllers that interact with each other and a continuous environment. Modeling and analysis of these types of *hybrid* systems is an active research area with many tools available for simulation and verification, such as HyTech [7], CheckMate [14], CHARON [1], d/dt [4], Teja [16], etc. They focus on the system-level dynamics and typically ignore real-time scheduling behavior of the embedded software on a shared processor during system analysis. There are also many tools based on real-time scheduling theory such as RapidRMA [18] and TimeWiz [17], but they do not represent the physical system explicitly. Rather, the physical environment is abstracted to a set of periodic or aperiodic interrupts that trigger sequences of tasks on the processor.

Traditionally, the control engineer designs the control algorithms without consideration of controller platform issues, and then hand them over to the software engineer, who implements them on a minimum cost controller platform while guaranteeing system schedulability for a set of task execution frequency requirements. The authors in [13, 5] propose to break the rigid wall between controller design and software implementation, and adopt an integrated approach, thus opening up the possibility of applying a range of offline optimization and online adaptation techniques. For example, instead of treating each task as having a rigid minimum execution frequency requirement of 40Hz, we can relax it to an interval of [35Hz, 40Hz]. The controller suffers performance degradation with slower execution frequencies as long as it still maintains critical control objectives such as system stability. This enables the designer to perform cost-performance tradeoff analysis.

In this paper, we propose an integrated approach for modeling and analysis of embedded real-time systems with tight coupling between *embedded* software and *embedding* physical environment, and analyze the real-time scheduling behavior of the software together with the physical system that the software is controlling within the same formal framework of *Hybrid Automata* (HA). We can also adopt other real-time formalisms such as variants of Petri-Nets with time [6] and Timed Automata [11], but as a formalism for describing hybrid systems, HA seems particular suitable since it accurately describes the hybrid nature of a computer-based control system. By adopting this approach, we enable the designer to have an integrated view of the entire system when making design decisions, so she can clearly see the effect of making a change in embedded software on the rest of the system, or a change in the physical system on embedded software design. She can also perform optimization analysis such as maximizing total system utility given resource constraints, or minimizing total system cost given safety and liveness requirements.

In order to model real-time behavior of embedded software, it is unavoidable to model processor contention due to multiple applications executing on the same platform. We can model real-time scheduling algorithms with a formal model such as HA, and use model checking to obtain end-to-end response time, thus enabling verification of software schedulability as well as system-level timing constraints within a single framework.

This paper is structured as follows. Section 2 provides a brief informal introduction to HA and the tool HyTech. Section 3 describes a generic approach for modeling fixed-priority scheduling with HA. Section 4 considers integrated modeling and analysis of the railroad crossing problem, and the paper concludes with Section 6.

2 Introduction to Hybrid Automata and Hytech

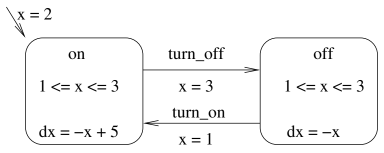

Figure 1: A Thermostat. The variable x denotes temperature; dx denotes 1st derivarive of x.

A *hybrid dynamical system* has both real-valued and boolean-valued variables. A system trajectory is a sequence of flows and jumps: during flows, the boolean part of the state stays constant and the real part of the state evolves over time; at jumps, the entire state changes instantaneously. We describe hybrid dynamical systems using *hybrid automata*(HA). A HA annotates the control graph of a finite automaton with conditions on real-valued variables. Each node of the graph represents an operating mode of the system, and is annotated with differential inequalities that prescribe the possible evolutions (flows) of the real variables while the system remains in the given mode. Each edge of the graph represents a switch in operating mode, and is annotated with a condition that prescribes the possible changes (jumps) of the real variables when the system executes the mode switch.

Figure 1 shows a HA model for a thermostat taken from [7]. It alternates between two operating modes: *on* or *off*. Initially, the heater is on and the temperature x is 2 degrees. When the heater is on, the temperature rises at the rate of $-x + 5$ degrees per minute; when the heater is off, the temperature falls at the rate of $-x$ degrees per minute. The heater can be turned off when the temperature reaches 3 degrees, and it can be turned on when the temperature falls to 1 degree. This is due to the *edge conditions* $x = 1$ and $x = 3$, which assert when a mode switch *may* occur. The *invariant conditions* $1 \leq x \leq 3$ of both operating modes prescribe that a mode switch *must* occur before the temperature leaves the operating interval of $[1, 3]$ degrees.

Hytech [7] is a model-checker for *Linear Hybrid Automata*(LHA), that is, the dynamics of the continuous variables are defined by *linear* differential inequalities. Since the state space for a hybrid system is infinite, Hytech performs exhaustive state space exploration symbolically (i.e., not enumeratively) by describing infinite state sets using linear constraints. A major strength of HyTech is its ability to perform *parametric analysis.* Often a system is described using parameters, and the designer is interested in knowing which values of the parameters are required for correctness. Hytech can be used to automatically synthesize the valid regions for the parameters.

3 Task Modeling with Hybrid Automata

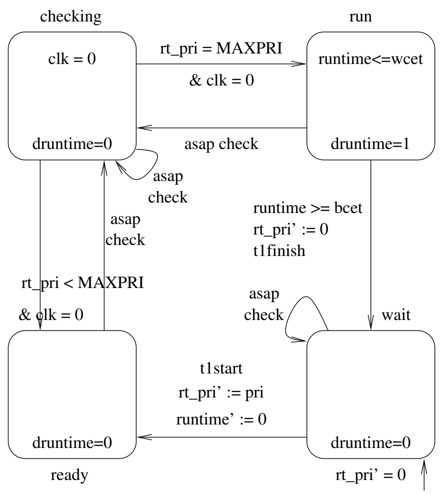

Figure 2: An event-triggered task scheduled under priority-based preemptive scheduling. A primed variable denotes the value of the variable after a discrete state change.

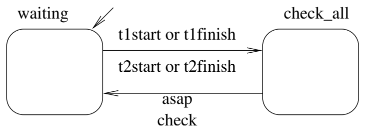

Figure 3: An auxiliary checker automaton that works with the task automaton in Figure 2. It assumes that there are 2 tasks in the system.

Figure 2 shows a event-triggered task modeled with HA, and Figure 3 shows a *checker* automaton that works together with the task automaton. Figure 4 shows a periodic timer that can be used to trigger the task automaton in Figure 2.

In order to model priority-based scheduling, it is necessary for the system to choose the highest priority task ready to run at any given moment. This only

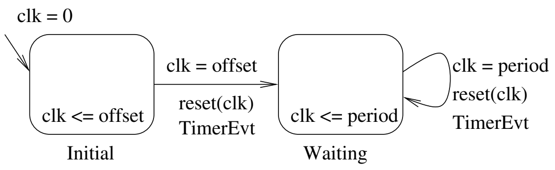

Figure 4: A periodic timer with *offset* and *period.* In order to be used with the task automaton in Figure 2, the name of the synchronizing event has to be the same in both automata, i.e., rename *TimerEvt* to *t1start*, or vice versa.

needs to be done at those *check instants* when a new task instance is triggered and ready to run, or a running task has just finished its execution. We model each real-time task with an automaton, in addition to a *checker* automaton in Figure 3, which issues a system-wide *check* signal at the *check instants* that forces each task automaton to check to see if it has the highest-priority at that moment.

A task automaton in Figure 2 contains several local variables:

- *pri* is a constant value denoting the task's *nominal* priority.
- *rt_pri* is a discrete variable denoting the task's runtime priority. It is set equal to the nominal priority *pri* when it is triggered and ready to execute, actively competing for the CPU; it is set equal to 0 when it has finished its execution and is inactive, waiting for the next activation trigger to arrive.
- *runtime* is a *stopwatch* variable that measures the amount of time that the task has spent in execution. In order to model preemptive scheduling, it is necessary to use a stopwatch instead of a *clock.*

A task initially goes into *wait* state with $rt\_pri = 0$, meaning that it is not ready to compete for the CPU yet. When it is triggered by a *t1start* event, which can be from a periodic timer or an aperiodic interrupt, it goes into *ready* state after setting $rt\_pri' = pri$, i.e., setting its runtime priority to its nominal priority. The *t1start* event also triggers the auxiliary automaton in Figure 3 to go from *waiting* to *check_all*. This enables the channel *check* and forces all the task automata currently in states *ready* or *run* into the *checking* state. Each task checks to see if it's the highest priority task. If yes, then it goes to the *run* state; if not, it goes to the *ready* state.[1] The auxiliary clock *clk*

[1]$rt\_pri = MAX\_PRI$ is a shorthand for a set of statements

ensures that the *checking* state is entered and exited instantaneously. *runtime* is a stopwatch that keeps track of the task's execution time by increasing at rate 1 in the *run* state, and not increasing in all other states. When the task has been running for at least *bcet* (best-case execution time), it can choose to finish its execution and go back to the *wait* state after issuing a $t1finish$ signal, which in turn triggers another round of check among all ready tasks. The task has to finish its execution before *wcet*(worst-case execution time), enforced by the invariant $runtime \leq wcet$ in location *run*.

In order to model non-preemptive scheduling, it is only necessary to remove the transition edge from the *run* state to *checking*, and add a self-transition from *run* state back to itself upon the signal *check*. This means that the task automaton ignores the *check* signal while it is running, i.e., it runs to completion once triggered.

An alternative modeling method [2] works by manually constructing the global scheduler automaton with states such as (assuming there are two tasks in the system, A and B) *AwaitBwait*, *AwaitBrun*, *ArunBwait* and *ArunBready*. This method does not scale up to more than a few tasks since the number of states grows exponentially with the number of tasks. Note that we are not solving the state space explosion problem that is a major limiting factor in the scalability of the model-checking approach; we are merely proposing a modeling approach that does not involve *manual* construction of an automaton with exponentially increasing size.

4 Railroad Crossing Problem

The Railroad Crossing (RC) problem, taken from [7], describes a railroad crossing, whose physical layout is shown in Figure 5, and whose behavior is given by the train/gate/controller automata in Figures 6, 7 and 8.

The system consists of 3 components: the train, the gate, and the controller. The train is initially some distance (at least 2000 meters) away from the track intersection with the gate fully raised. As the train approaches, it triggers a sensor 1000 meters ahead of the intersection, signaling its upcoming entry to the controller. Upon receiving the signal, the controller performs some internal computation that causes a delay between [$bcet\_lower, wcet\_lower$] seconds, then sends

comparing the current task's priority to those of the other tasks in the system. Also note that for the sake of brevity, the current model formulation does not deal with tasks with equal priority value. This can be addressed by assigning an index number to tasks with equal priority in order to break the tie.

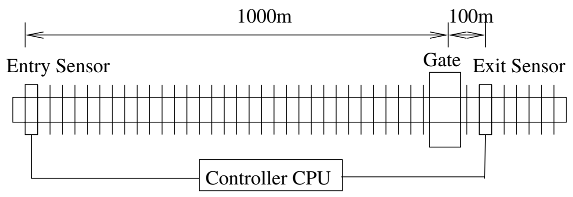

Figure 5: Railroad Crossing System.

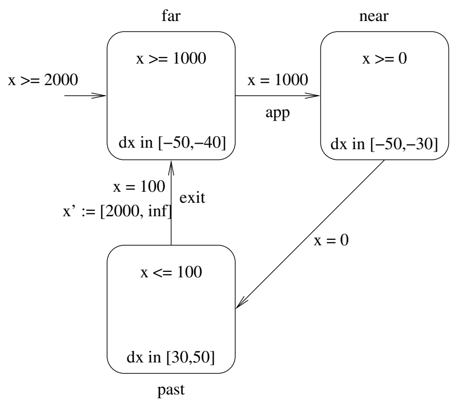

Figure 6: The train automaton. Variable x is the distance of the train from the gate.

a *lower* command to the gate. When the gate receives a lower command, it lowers itself at a rate of 9 degrees per second. After the train has exited the intersection and is 100 meters away, It sends an *exit* signal to the controller. After a computational delay between [$bcet\_raise, wcet\_raise$] seconds, the controller commands the gate to be raised. Note that the delay from triggering of the enter/exit sensors to issuing the lower/raise command to the gate is likely not caused by computational activities, since the computation is trivial, but rather system design parameters implemented by a time out mechanism. For our purposes we assume it's computational delay in order to introduce processor contention later.

The system has to satisfy two properties:

- *Safety*: Whenever the train is in the crossing, the gate has to be lowered.
- *Bounded liveness*: Within a certain time limit δt after the train leaves the crossing, the gate has to be raised.

Although the Railroad Crossing problem is a standard textbook problem in real-time specification and verification, there has been little discussion about the

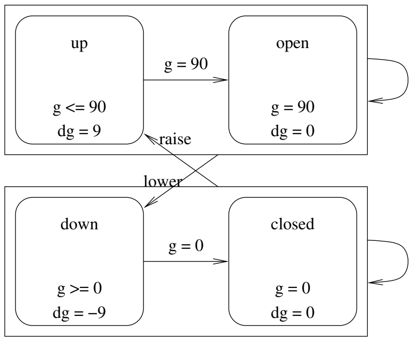

Figure 7: The gate automaton. Variable g is the angle of the gate relative to the horizontal position. When $g = 0$, the gate is lowered; when $g = 90$, the gate is raised.

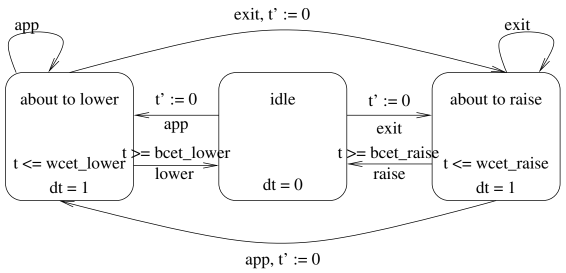

Figure 8: The controller automaton. It is a event-driven task triggered by sensor signals *app* and *exit*. The *app* signal triggers the task T_{lower}, and the *exit* signal triggers the task T_{raise}. Since this is the only task running on the CPU, there is no processor contention, and each task runs to completion.

real-time scheduling behavior of the controller computer. That is, it is generally assumed that the controller is dedicated to a single task with no interference from higher-priority tasks or operating systems activity, so there is no need for analyzing the real-time scheduling behavior of the controller software. This may well be true for the simple controller we are considering here, but in general designers have been putting more and more functionality on a single micro-controller in order to reduce costs. Furthermore, there is also a tendency to take advantage of distributed, multi-processor platforms. In these kinds of complex embedded systems, the real-time scheduling problem is non-trivial to solve, and it is desirable to model the scheduling and runtime platform issues explicitly.

In order to make the problem more interesting, we consider several parallel train tracks, each having a separate gate. A single controller CPU is used to control all of the train tracks. The controller tasks are logically independent, since the tracks are independent, but they interfere with each other due to contention for the shared CPU resource.

Figure 9 shows a controller model that explicitly deals with real-time scheduling issues. Compare it to Figure 8, the states *about to lower* and *about to raise* have been expanded into multiple states based on real-time task modeling approach in Figure 2. We can view the controller software as a task-chain consisting of a two tasks: the *lower-gate* task T_{lower} trigged by the entry sensor, followed by the *raise-gate* task T_{raise} triggered by the exit sensor. A back-of-the envelop calculation shows that the time interval between the two task triggers lies between $[22, 61.6]$ seconds. Since the distance between the entry sensor and exit sensor is $1100m$, and the maximum speed of the train when it is travelling between the two sensors is 50m/s, the minimum amount of time between triggers of the two sensors is $1100/50 = 22$s. The upper bound of 61.6 can be calculated similarly. We assign a higher priority to T_{lower} than T_{raise}, since the former affects the safety property, which is obviously more important than the bounded liveness property. That is, we are willing to suffer delays in raising the gate in order to ensure that the gate is lowered in time for the train to pass. The priority assignment does not matter if there is only one task chain, since T_{lower} and T_{raise} are precedence-constrained and do not interfere with each other, but it does matter when there are multiple task chains running on the same processor.

This task model does not fit the assumptions of Rate Monotonic Analysis (RMA) [8], and would require non-trivial extensions to RMA in order to determine its schedulability. It is also intimately tied to the physical environment, which provides the task triggers in a non-periodic fashion. By modeling the task scheduling behavior explicitly together with the physical environment, we can use model-checking to determine schedulability as well as verify system-level safety and bounded liveness properties within the same framework.

Let's assume there are 3 train/gate combinations, hence 3 task chains running on a single controller CPU. All 3 set of trains and gates have the same timing parameters specified in Figure 6 and Figure 7. The controller task chains have timing specifications in Table 1.

HyTech reveals that the system does not satisfy the safety requirement, and produces a counter-example leading to the safety violation. Essentially, when all 3 trains arrive at the same time, task T_{lower3} has the lowest priority, so it does not finish its execution and

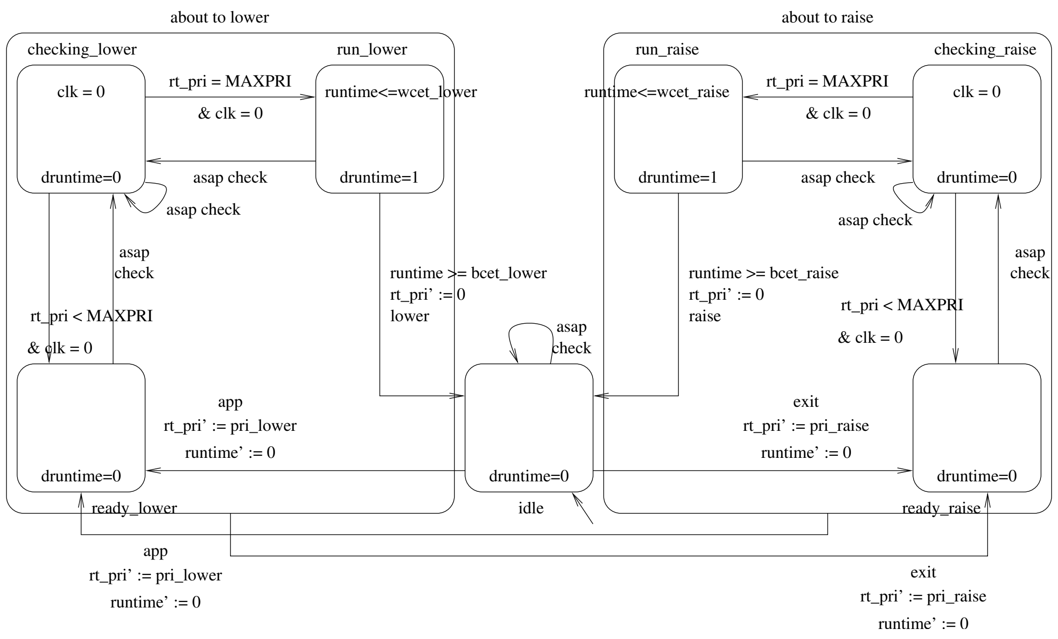

Figure 9: The Controller in the presence of processor contention. We augment the model in Figure 8 with real-time scheduling information, since there are multiple controller tasks running on the same CPU. The scheduling discipline is preemptive fixed-priority.

	T_{lower1}	T_{lower2}	T_{lower3}
Priority	6	5	4
[bcet,wcet]	[2.0,3.0]	[3.0,4.0]	[3.0,4.0]
	T_{raise1}	T_{raise2}	T_{raise3}
Priority	3	2	1
[bcet,wcet]	[3.0,4.0]	[3.0,4.0]	[3.0,4.0]

Table 1: The timing parameters for the 3 controller task chains. Larger priority value denotes higher priority. Time units are in seconds.

issue a *lower* signal to the gate until 11s after the train arrival. The gate takes 10s to lower itself. Meanwhile, the train travels at maximum speed of 50m/s, and arrives at the crossing 20s after tripping the entry sensor, when the gate is not yet fully lowered.

We can take advantage of the *parametric analysis* capability of HyTech to derive certain timing parameters, whether those of the software or the physical environment, given timing specification for the rest of the system, in order to satisfy system-level requirements. For example, we can set T_{lower3} as a parameter, and ask the question: what is the range of values for T_{lower3} such that the system safety requirement is satisfied? Similarly, we can ask: if we cannot change the timing properties of the controller tasks, what is the maximum speed of the train between tripping the entry sensor and arriving at the gate? Using parametric analysis, the designer can come up with a number of options to remove the safety violation:

- Switch to a faster execution platform, and reduce T_{lower3} to be below 3s.
- Add more controller CPUs to the system, and run each controller task on a separate CPU to improve system responsiveness.
- Impose a reduced speed limit on incoming trains once they reach the entry sensor position, so that the minimum time the train takes to reach the crossing from the entry sensor position is above 21s.
- Switch to a more responsive gate so that the time it takes to raise or lower the gate is below 9s.

Of course we can adopt a combination of any subset of the above options with relaxed parameters for each individual option.

Concerning the bounded liveness property, parametric analysis reveals that the gate is raised at a minimum of $\delta t = 24s$ after the train crosses the gate. We can also impose a more strict requirement, say

$\delta t \leq 18s$, and synthesize various parameters of the system that satisfies the requirement.

5 Related Work

Model-Integrated Computing (MIC) [15] uses integrated, multi-view, domain-specific models to capture information relevant to the system under design. Models can represent the designer's understanding of an entire computer-based system, including information-processing architecture, physical architecture, and operating environment. The Generic Modeling Environment (GME) [10] explicitly represents dependencies and constraints among various modeling views, and can be used for generating system implementation as well as specialized models that feed into various analysis tools such as model-checkers. Our approach can be viewed as a specific instance of the more general concept of MIC, using a formal model (Hybrid Automata) and analysis method (model-checking), while MIC allows the designer to uses meta-modeling techniques to construct arbitrary domain-specific modeling environments. In fact we can construct a meta-model for HA within GME, synthesize a graphical modeling environment that acts as a front-end to the user, and write an interpreter to generate the textual input to the HyTech model-checker, which lacks a graphical user interface. We can also imagine providing the designer with an even higher level of abstraction by using an UML-like modeling notation within GME, enhanced with real-time and hybrid system concepts, and generate HyTech models automatically. This approach allows the user to use a familiar industry-standard notation instead of a formal modeling language.

Seto [13] proposed an integrated approach to controller design and task scheduling, where task frequencies are allowed to vary within a certain range as long as such a change does not affect critical control functions such as maintenance of system stability. An algorithm was proposed that optimizes the overall system control performance while maintaining schedulability by adjusting task frequencies. Similarly, Eker [5] presented a Matlab toolbox for simulation of a real-time kernel in parallel with continuous plant dynamics. The toolbox allows the user to study the interactions between the control tasks and the scheduler, making it possible to experiment with more flexible approaches to real-time control systems, such as feedback scheduling. This body of work deals with controller performance with traditional metrics in control theory such as signal rise time, stability, etc., while our focus is on modeling of system real-time behavior, and static, offline verification through model-checking, although it is possible to take advantage of other existing hybrid modeling tools for integrated simulation(HyTech does not provide a simulation component).

Norstrom [11] extended the classic model of Timed Automata(TA) with a notion of real-time tasks. A discrete transition in an extended TA denotes an event releasing a task and the guard on the transition specifies all the possible arriving times of the event. The schedulability problem can be transformed into a reachability problem for TA. Our work is similar in that we model real-time scheduling together with the physical environment. Our use of HA instead of TA enables more accurate modeling of the physical system. Other approaches to formal modeling and analysis of real-time scheduling include Corbett's work on timing analysis of Ada tasking programs [3], and Lee's work on real-time process algebra ACSR-VP [9]. None of them proposes an integrated modeling approach.

6 Conclusions and Future Work

In this paper we have proposed an integrated approach for modeling and analysis of embedded real-time systems with tight coupling between *embedded* software and *embedding* physical environment, where the physical system and the software artifacts are modeled within the same formal framework. We have also described a generic method for modeling fixed-priority scheduling with Hybrid Automata. Our approach allows the designer to model and analyze the embedded system in an integrated manner, including the physical system and the software controlling it, and use model-checking to determine schedulability of the software together with system-level timing constraints.

Although we have used a specific modeling formalism (HA), this approach is generic and can be applied together with other formal or informal models commonly used in the embedded systems domain, such as timed or hybrid variants of Petri-Nets [12]. We have used a simple example, railroad crossing, to illustrate our approach. It is our intention and ongoing work to model and analyze a more realistic system that involves non-trivial hybrid behavior in both the physical system and software. We are also considering using other hybrid analysis tools directly in order to avoid some of HyTech's limitations, such as restriction to linear dynamics, lack of structural/behavioral hierarchy and graphical user interface, lack of simulation capability, etc.

References

[1] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pap-

pas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In *Proceedings of EMSOFT'01:First Workshop on Embedded Software*, 2001.

[2] Steven Bradley, William Henderson, and David Kendall. Reducing conservatism in response time analysis of distributed systems. In *Proceedings of the IEE Colloquium on Real-Time Systems*, 1999.

[3] James C. Corbett. Timing analysis of Ada tasking programs. *IEEE Transactions on Software Engineering*, 22(7):461–483, July 1996.

[4] T. Dang and O. Maler. Reachability analysis via face lifting. *Hybrid Systems: Computation and Control. LNCS*, 1386, 1998.

[5] J. Eker and A Cervin. A matlab toolbox for real-time and control systems co-design. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, pages 320 –327, 1999.

[6] Robert Esser, Jrn W. Janneck, and Martin Naedele. Applying an object-oriented petri net language to heterogeneous systems design. In *Proceedings of Petri-Nets in System Engineering*, 1997.

[7] T. Henzinger, P. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer, special issue on timed and hybrid systems*, pages 110–112, 1997.

[8] Mark H. Klein, Thomas Ralya, Bill Pollak, and Ray Obenza. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[9] Hee-Hwan Kwak, Insup Lee, Anna Philippou, Jin-Young Choi, and Oleg Sokolsky. Symbolic schedulability analysis of real-time systems. In *RTSS*, pages 409–, 1998.

[10] Akos Ledeczi, Miklos Maroti, Arpad Bakay, and Gabor Karsai. The generic modeling environment. In *Proceedings of the IEEE International Workshop on Intelligent Signal Processing*, May 2001.

[11] C. Norstrom, A. Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 182–189, December 1999.

[12] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. In *Technical Report TR 120, Massachussets Institute of Technology*, February 1974.

[13] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin. On task schedulability in real-time control systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 13–21, 1996.

[14] B. Silva, K. Richeson, B. Krogh, and A. Chutinan. Modeling and verifying hybrid dynamic systems using checkmate. In *Proceedings of the 4th International Conference on Automation of Mixed Processes*, 2000.

[15] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *IEEE Computer*, 30(4):110–111, April 1997.

[16] Teja Technologies website. http://www.teja.com.

[17] TimeSys website. http://www.timesys.com.

[18] Tripacific Software website. http://www.tripac.com.