99PC-314

A Hybrid System Solution Of The Interrupt Latency Compatibility Problem

Feng Lin
Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202

David T. Ashley
Electronics Technical Center, Visteon Automotive Systems, 17000 Rotunda Drive, Dearborn, MI 48121-6010

Michael Heymann
Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel

Michael J. Burke
Electronics Technical Center, Visteon Automotive Systems, 17000 Rotunda Drive, Dearborn, MI 48121-6010

Copyright © 1997 Society of Automotive Engineers, Inc.

ABSTRACT

Microprocessors and microcontrollers are now widely used in automobiles. Microprocessor systems contain sources of interrupt and interrupt service routines, which are software components executed in response to the assertion of an interrupt in hardware. A major problem in designing the software of microprocessor systems is the analytical treatment of interrupt latency. Because multiple interrupt service routines are executed on the same CPU, they compete for the CPU and interfere with each other's latency requirements. Here, interrupt latency is defined as the delay between the assertion of the interrupt in hardware and the start of execution of the associated interrupt service routine. It is estimated that 80% of intermittent bugs in small microprocessor software loads are due to improper treatment of interrupts. Until this work, there is no analytic method for analyzing a particular system to determine if it may violate interrupt latency requirements. There is also no reliable empirical method for ruling out the possibility of interrupt latency violations in a particular system, as they may occur under only very specific conditions. We use a newly developed hybrid system approach to solve this interrupt latency compatibility analysis problem. We have developed an efficient algorithm to determine if interrupt latency violations may occur in a particular system. A software tool that implements the algorithm is also being developed. With such software, we can easily check if interrupt latency constraints may be violated under any circumstances. If so, such software may also indicate how to modify the interrupts and interrupt service routines to avoid such violations.

INTRODUCTION

Microprocessor systems contain interrupt sources (*IS*s), which can request service from the processor (assert an interrupt), and must always have a software response (the execution of the corresponding software interrupt service routine, or *ISR*) starting within an interval of time (the maximum allowable interrupt latency, or *AIL*) after the assertion of the hardware interrupt. Because a processor may execute only one stream of instructions at a time, the execution of one ISR may delay the execution of another ISR until the first has completed. Hence, ISs and their associated ISRs interfere with each other's interrupt latency requirements and may not be combined arbitrarily in the same microprocessor system without the possibility of the AIL being violated for some or all ISs.

As a necessary condition to guarantee that a microprocessor system will always behave as intended, each IS must always be serviced by its corresponding ISR within the AIL. Verifying that the interrupt latency requirement is always satisfied for every IS is the interrupt latency compatibility problem that we address in this paper.

The importance of the interrupt latency compatibility problem in microprocessor system design cannot be overstated. A system accidentally constructed so that interrupt latency incompatibilities exist has a high probability of exhibiting serious behavioral defects which are *intermittent*, because the conditions required to

cause an interrupt latency violation may involve specific timing relationships between the assertion of different hardware interrupts. Furthermore, it is uncertain that product testing will excite the system with these necessary timing relationships. For an automobile manufacturer which produces microprocessor systems with firmware that cannot be altered after production, intermittent defects may cause disappointed customers, warranty returns, or government-mandated vehicle recalls. For systems which may cause injury or loss of life through defective behavior—such as airbag deployment systems, ABS systems, throttle controls, or fly-by-wire controls—the importance of minimizing the probability of intermittent defective behavior is clear.

The interrupt latency compatibility problem is difficult to solve, and has defied an analytical solution by the mathematical and computer science community for more than 25 years[1]. We believe that the reason this problem has defied solution is that it combines both discrete and continuous elements, and a mathematical framework in which this problem can be embedded (hybrid systems) is a relatively recent development.

To analytically verify that interrupt latency requirements can *never* be violated, it must be verified that they will never be violated in *any* possible run of the system. The complexity of practical systems makes it impossible to exhaustively search all possible runs of the system. For the same reason, testing or simulation results can never be conclusive.

In the view of the authors, the only practical way to solve the interrupt latency compatibility problem is by an intelligent *analytical* approach that will consider all possible runs of the system without an exhaustive search. We believe that such an approach exists using a theory that we have developed recently for a class of so-called *hybrid systems*.

Hybrid is used to classify a system consisting of both continuous dynamics and discrete events. In our context, continuous dynamics describe essential continuous elements of the system such as the passage of time, while discrete events describe events such as the assertion of an interrupt or a discrete state change in digital hardware. Both continuous dynamics and discrete events are needed in describing or modeling the interrupt latency compatibility problem.

To implement our solution, we model each IS by an elementary hybrid machine (*EHM*). An EHM consists of several configurations that describe the status of the IS, such as *Interrupt Not Pending*, *Interrupt Pending*, etc. (Figure 3). Transitions from one configuration to another are also specified. We also model the software system (*SS*) as an EHM to reflect that generally only one ISR may run at a time, and to capture the interaction between the ISRs and the ISs (Figure 2). Then, a well-defined parallel composition of all EHMs can be used to describe the concurrent operation of several ISs and ISRs, resulting in a composite hybrid machine (*CHM*).

After obtaining a CHM model of the overall system, violation of interrupt latency requirements can now be specified by identifying all illegal configurations. Therefore, the question of whether interrupt latency requirements can be violated is equivalent to the question of whether any of the illegal CHM configurations can be reached from the initial configuration. Verification is now reduced to reachability analysis.

Reachability analysis of hybrid systems is a difficult problem. In this paper, we develop an efficient algorithm to determine reachability for a class of hybrid systems encountered in the interrupt latency compatibility problem. Our algorithm uses a *logical* approach by manipulating logical conditions involving the timing of events. We find this logical approach to be excellent in the specification and implementation of reachability analysis.

Using our approach, a necessary and sufficient condition for meeting the interrupt latency requirements is obtained. The interrupt latency requirements can be violated ***if and only if*** some illegal configurations are reachable from the initial configuration. Therefore, there is no conservatism in our approach. This is important because conservatism may add unnecessary cost to a satisfactory system by misclassifying it as capable of violating its interrupt latency requirements, thus forcing unnecessary increases in product cost.

We are aware of several proposed approaches to the interrupt latency compatibility problem which have appeared in the literature dating back to at least the early 1970s. Among the existing approaches, rate-monotonic analysis (RMA) [5] is best-known and most widely used. Although RMA is well-developed and has been used extensively in practice, there are difficulties in applying the mathematical results of RMA to the interrupt latency compatibility problem. First, RMA can only be used for perfectly periodic (i.e. rate-monotonic) ISs. This is a serious limitation, as most interrupts in automotive systems are not periodic. Second, RMA gives only sufficient conditions for satisfying interrupt latency requirements. Therefore, a satisfactory system may be erroneously classified by RMA as able to violate its interrupt latency requirements, provoking unnecessary cost increases. Finally, RMA can be applied only to fixed priority preemptive scheduling.

The approach we propose has additional advantages. One advantage is the ability to model the discrete behaviors of a system, which cannot be modeled or analyzed using RMA or other approaches which rely on continuous mathematics alone. We have found that we

[1] We believe that computer processors relying on hardware interrupts were common no later than 1972.

can use our approach to model ISRs which vary their execution time in cyclical patterns (typical of ISRs associated with periodic ISs). We are also able to model queued communication peripherals and the associated ISRs, where the amount of ISR execution time required to retrieve data from the peripheral may increase with increasing interrupt latency. Finally, we are able to model non-atomic ISRs (ISRs that may be interrupted).

Since the parallel composition of EHMs and the reachability analysis must in practice both be performed automatically by a computer, we offer some insight into the construction and availability of software tools to perform these tasks.

MATHEMATICS OF HYBRID MACHINES

In this section, we present the hybrid machine model developed in [6] to specify the interrupt latency compatibility problem. Using this model, each process (e.g., each SS or IS) is modeled by an elementary hybrid machine denoted by

$$EHM = (Q,\Sigma, D, I, E, (q_0, x_0)).$$

The elements of an EHM are

1. Q, a finite set of vertices.

2. Σ, a finite set of event labels. An event is an input event, denoted by $\underline{\sigma}$ (underscore), if it is received by the EHM from its environment; and an output event, denoted by σ, if it is generated by the EHM and transmitted to the environment. [2]

3. $D = \{d_q = (x_q, y_q, u_q, f_q, h_q): q\in Q\}$, the dynamics of the EHM, where d_q, the dynamics at the vertex q, is given by:

   $$x'_q = f_q(x_q, u_q),$$

   $$y_q = h_q(x_q, u_q),$$

   with x_q, u_q, and y_q, respectively, the state, input, and output variables of appropriate dimensions. For the interrupt latency compatibility problem, we consider only x'_q=*constant*[3] and $y_q = x_q$. Therefore, we need not distinguish the output and state variables. (A vertex need not have dynamics associated with it, that is $d_q = \emptyset$, in which case we say the vertex is static.)

4. $I = \{I_q : q\in Q\}$, a set of invariants. I_q represents conditions under which the EHM is permitted to reside at q. Formally, an invariant is a predicate which is defined as a Boolean combination of inequalities (called atomic formulas) of the form

   $$s_i \geq C_i \text{ or } s_i \leq C_i,$$

   where s_i is a shared continuous real variable (or *SV*, to be defined soon), and C_i is a real constant.

5. $E = \{(q,G\wedge\underline{\sigma}\rightarrow\sigma',q', x_{q'}^0): q, q'\in Q\}$, a set of edges or transitions, where q is the exited vertex, q' the entered vertex, $\underline{\sigma}$ the input event, σ' the output event, G a predicate having the same form as invariants I_q, and $x_{q'}^0$ the initialization value for $x_{q'}$ upon entry to q'. We allow the initialization to take on one of three forms.

   (a) $x_{q'}^0$=*constant*;

   (b) $x_{q'}^0 = g(x_q)$, that is, $x_{q'}^0$ is a function of x_q; or

   (c) $x_{q'}^0 = \text{nondet}(v_l, v_u)$, that is, $x_{q'}^0$ takes a value between v_l and v_u.[4]

   An edge $(q,G\wedge\underline{\sigma}\rightarrow\sigma',q', x_{q'}^0)$ is interpreted as follows: if G is true and the event $\underline{\sigma}$ is received as an input, then the transition to q' occurs with the assignment of the initial condition $x_{q'}^0(t_0) = x_{q'}^0$. Here t_0 denotes the time at which the vertex q' is entered. The output event σ' is transmitted at the same time.

   Although a more general model was presented in [6], in this paper we assume that when the EHM enters vertex q, its invariant I_q must be satisfied. Thus, let us define $wp(q,G\wedge\underline{\sigma},q')$ to the weakest precondition under which the transition $(q,G\wedge\underline{\sigma},q')$ will not violate the invariant I_q upon entry to q. Since some of variables that appear in I_q are possibly (re-)initialized upon entering q, the condition $wp(q,G\wedge\underline{\sigma},q')$ can be calculated from I_q by substituting into I_q the appropriate initial values. That is, if $x_{q'}$ is (re-)initialized to $x_{q'}^0$, then

   $$wp(q,G\wedge\underline{\sigma},q') = I_q|_{x_{q'} = x_{q'}^0}.$$

   Since we allow nondeterministic (re-)initialization, care must be taken when we calculate $wp(q,G\wedge\underline{\sigma},q')$. For instance, if $I_q=[s_i \geq C_i]$ and $s_i^o = \text{nondet}(k_l,k_u)$, then

[2] The environment of an EHM includes all other EHMs, hence an event will be denoted σ in the generating EHM and $\underline{\sigma}$ in the EHM which accepts this event as input.

[3] For our models, this constant is usually 1 (to allow a continuous variable to mimic elapsed time).

[4] *nondet*(v_l, v_u) does not imply a probability distribution, it denotes that the value assigned has some uncertainty but will be in $[v_l, v_u]$. The reachability algorithm we have developed correctly treats such assignments.

$$wp(q, G \wedge \underline{\sigma}, q') = \begin{cases} true & if\ k_l \geq C_i \\ false & otherwise \end{cases}$$

This is because if $k_l \geq C_i$, then in the worst case $[s_i \geq C_i]$ is true, so that the transition (q, G∧σ, q') will not violate the invariant I_q'. If the weakest precondition is not satisfied, that is, G does not imply wp(q,G∧σ,q'), then we will replace (q,G∧σ→σ',q', $x_{q'}^0$) by (q, wp(q,G∧σ,q') ∧G∧σ→σ',q', $x_{q'}^0$).

If σ' is absent, then no output event is transmitted. If $x_{q'}^0$ is absent, then the initial condition is inherited or partially inherited from x_q (assuming the inherited states represent the same physical object and hence are of the same dimension). If σ is absent, then the transition takes place immediately upon G becoming true. Such transitions will be called *dynamic* transitions. If G is absent, the guard is always true and the transition will be triggered by the input event σ. Such transitions will be called *event* transitions.

Although our EHM model allows guarded event transitions with both G and σ, such transitions can be decomposed into event transitions and dynamic transitions as shown in Figure 1.

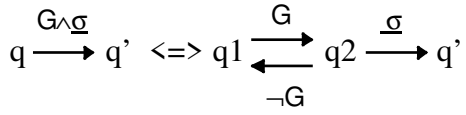

Figure 1: Guarded Event Transition

In the figure, q has been partitioned into q1 and q2, with $I_{q\ 1} = I_q \wedge \neg G$ and $I_{q\ 2} = I_q \wedge G$. This decomposition is valid because for a guarded event transition to take place, the guard must be true when the event is triggered.

6. (q_0, x_0) denotes the initialization condition: q_0 is the initial vertex and $x_{q0}(0) = x_0$.

Without loss of generality, we assume that the invariant I_q is violated if and only if one of the guards at q is satisfied, that is,

$$I_q = \neg G_1 \wedge \neg G_2 \wedge \ldots \wedge \neg G_k,$$

where $G_1, G_2, \ldots, G_k$ are guards of the dynamic transitions leaving q. If this assumption is not satisfied, we can simply redefine I_q as above without changing the EHM.

When several processes modeled by EHMs run in parallel, the concurrent system will be modeled by a composite hybrid machine (CHM) using a parallel composition operator ||.

$$CHM = EHM^1||EHM^2||...||EHM^n.$$

Interaction between EHMs is achieved by means of signal transmission (shared variables, or *SV*s) and input-output event synchronization (message passing or event passing) as described below.

Shared variables consist of output signals from all EHMs as well as signals received from the environment. They are shared by all EHMs in the sense that they are accessible to all EHMs. A specific shared variable s_i can be the output of at most one EHM. The set of shared variables defines a signal space $S = [s_1, s_2, \ldots, s_m] \in R^m$.

Transitions are synchronized by an input-output synchronization formalism. That is, if an output event σ is either generated by one of the EHMs or received from the environment, then all EHMs for which σ is an active transition label (i.e., σ is defined at the current vertex with the guard satisfied) will execute σ (and its associated transition) concurrently with the occurrence of σ. A specific output event can be generated by at most one EHM.

Formally, a CHM is defined as follows.

$$CHM = EHM^1||EHM^2||...||EHM^n = (Q, \Sigma, D, I, E, (q_0, x_0)),$$

where

1. $Q = Q^1 \times Q^2 \times \ldots \times Q^n$ is the set of configurations;
2. $\Sigma = \Sigma^1 \cup \Sigma^2 \cup \ldots \cup \Sigma^n$ is the set of events;
3. $D = \{d_q = (x_q, y_q, u_q, f_q, h_q): q = <q^1, q^2, \ldots, q^n> \in Q\}$ combines all the dynamics of $q^1, q^2, \ldots, q^n$;
4. $I = \{ I_q = I_{q1} \wedge I_{q2} \wedge \ldots \wedge I_{qn}: q = <q^1, q^2, \ldots, q^n> \in Q\}$ is the invariant;
5. E is the set of edges or transitions, described below.
6. $(q_0, x_0) = (<q_0^1, q_0^2, \ldots, q_0^n>, (x_0^1, x_0^2, \ldots, x_0^n))$ is the initial condition.

The transitions in the CHM can also be derived from the EHMs. In general, the transitions can either be triggered by an input event from the environment or by a dynamic transition in one of the EHMs. Note that one transition in an EHM can trigger other transitions in other EHMs.[5] Therefore, a transition in the CHM may consist of a logically triggered finite chain of transitions in the EHMs. These transitions are considered to occur instantaneously and concurrent vertex changes in the EHMs occur at exactly the same instant.

For the interrupt latency compatibility problem, however, we can assume that the system is closed or self-

[5] We assume that there is no infinite chain of transitions, as this will lead to a Zeno system (see [6]).

contained[6]. Therefore transitions in the CHM are always triggered by dynamics in one of the EHMs. We denote these transitions by a triple

$$(q,G,q') = (<q^1, q^2, \ldots, q^n>, G, <q'^1, q'^2, \ldots, q'^n>)$$

where $q=<q^1, q^2, \ldots, q^n>$ is the source configuration, $q'=<q'^1, q'^2, \ldots, q'^n>$ the target configuration, and G is a guard. To have such a transition, there must exist a transition $(q^m, G \rightarrow \sigma', q'^m, x_{q'^m}{}^0)$ in some EHM^m, that triggers the transition. Note that for simplicity, we do not write the output events and initial conditions, they are inferred from the EHMs.

We use EHMs to model ISs and ISRs. The overall system is then described by the CHM. Since the above construction of a CHM from EHMs can be implemented automatically by a computer, a designer need only specify the EHMs. For the systems we have considered, the number of vertices in EHMs describing ISs is usually between 5 and 10. The EHM describing the SS may contain far more vertices, but there is only one such EHM in a system.

To specify interrupt latency requirements, we identify an illegal vertex in each EHM model of an IS. In other words, when the exhibited interrupt latency exceeds a given bound, the EHM will enter the vertex labeled *illegal* via a dynamic transition guarded by G=[t>b], where *t* is the waiting time and *b* is the given bound. A configuration in the CHM is illegal if at least one of its vertices is illegal. The set of illegal configurations is denoted by Q_b. Clearly, the interrupt latency requirements will be violated in some runs of the system if and only if at least one illegal configuration can be reached from the initial configuration.

MODELING OF MICROPROCESSOR SYSTEMS

This section outlines the modeling of microprocessor systems. The goal of modeling is to model the system as a collection of EHMs, which can then be combined to obtain a CHM, and the CHM can be checked for the reachability of illegal configurations. An EHM timing model of a microprocessor system with ISs and ISRs must involve separate EHMs for each IS as well as a separate EHM for the SS. The reason for this modeling requirement is that, except for certain events which force synchronization between separate EHMs, each IS and the SS may change state independently. An interrupt source (such as a UART) may change state in response to a significant event (such as a received character or the passage of time) without affecting other ISs or the SS. Software, of course, may change state without affecting any IS.

Before we describe modeling in detail, we need to clarify two subtle modeling points. First, the AIL is not a constant (it varies with the state of the system); and second, the AIL will not appear explicitly in any EHM.

Up to this point, we have described the AIL as if it were a constant. In practice, the AIL will change with the state of the system. Consider a free-running 16-bit counter (i.e. a timer) which is interfaced to the SS so that it generates an interrupt and causes execution of the same ISR (a)when the counter reaches the value of $F000_{16}$; and (b)when the counter rolls over from $FFFF_{16}$ to 0000_{16}. Depending on how this IS is used in the system, there may be the requirement that the ISR complete before the hardware interrupt is asserted again. In this example, this would imply that half of the time when the interrupt is asserted, a relatively short AIL is involved (on the order of $0FFF_{16}$ counts of the free-running counter); but that the other half of the time, a relatively long AIL is involved (on the order of $F000_{16}$ counts of the free-running counter). Furthermore, which of the two AIL values is appropriate is not random—the AIL alternates in a perfectly reliable A-B-A-B-… pattern.

The key salient point about the timer example above is that *the patterns of interrupts which have occurred in the past place restrictions on (and hence supply information about) the patterns of interrupts which may occur in the future*. The timer example is by no means contrived or unique in this regard. In our modeling work, we have found qualitatively similar examples involving UARTs, timers, and network communication peripherals. We should further add that this phenomenon is not restricted to one IS at a time—certain forms of “collusion” between different ISs are also possible. Consider two network communication peripherals (on the same network) which are interfaced to the same microprocessor. Assume that each of the two communication peripherals will only receive a certain set of network messages, and that the two message sets are mutually exclusive. If the physical characteristics of the network are such that only one message can be transmitted at a time, a receive interrupt asserted by one of the two communication peripherals places strong restrictions on when in time a receive interrupt may be asserted by the other communication peripheral. Similar collusion may exist between transmit and receive interrupts in the same communication peripheral.

The strength of our approach is that it can consider the interrupt history of the system (if the system is modeled well), and thus can treat an AIL which is state-dependent. We are able to model ISs where past patterns of interrupts place restrictions on future patterns

[6] This assumption significantly simplifies the reachability algorithm to be presented, although we do have a second and more complex algorithm that can handle open systems. Note that this assumption is made without essential loss of generality, because we can model the environment by an additional EHM that triggers the event transitions at an arbitrary time by using *nondet()*.

of interrupts, and systems where ISs collude. Our approach represents a necessary-and-sufficient test of any system that we can model in our framework, rather than a sufficient test.

It should also be noted that in practice the AIL will never appear explicitly in any EHM. We have defined the AIL to be the maximum allowable time between the assertion of an interrupt by an IS and the start of execution of the associated ISR. EHM IS models simply contain transitions to the illegal configuration (*ILC*). In principle it would be possible to manually determine the AIL from the EHM models by starting at the IS ILC and working backwards while simultaneously examining the ISR EHM and the interactions between the IS and the ISR, in order to determine the latest relative time the ISR can start while avoiding the ILC. However, we have never done this and we suspect it would be tedious for complex models.

OVERVIEW OF MODELING SOFTWARE SYSTEMS

Figure 2 shows the EHM timing model of a software system (*SS*). The system has one RTPI (ready to process interrupts) configuration, which models the SS as it is executing instructions and is able to be interrupted. ISs use an SV as an interface to the SS to indicate that they require service; we call this SV the RSSV (*r*equesting *s*ervice SV). At any time when no ISs are asserting an RSSV, the SS may enter a critical section (CS), which is a series of instructions executed with interrupts disabled. During a critical section, the software may not enter an ISR. Once the SS begins executing an ISR, another ISR may not begin execution until the SS returns to the RTPI configuration. We refer to an ISR which must run to completion as an *atomic* ISR. Non-atomic ISRs are treated as an exceptional case and discussed seperately.

Figure 2 shows that a CS may be entered at any time when no IS is asserting its RSSV. The guard on entry to the CS will reflect this aspect of a practical system—microprocessor hardware is almost always designed so that if multiple interrupts are pending, no non-ISR instruction will be executed until no interrupt is pending. In the figure, the CS guard contains an event $\underline{\sigma}$, which is an input event from the environment. This denotes that the CS may or may not be entered when no IS RSSV is pending. In other words, $\underline{\sigma}$ is a shorthand for an event which may occur at an arbitrary time.

For our modeling, we assume that the execution time of the CS may be in a certain range, with the lower bound derived from the shortest CS in the SS, and the upper bound derived from the longest CS in the SS. We are not confident that it is adequate to model the CS as the longest CS in the SS (that is, replacing *t=nondet(L,U)* by *t=U*), as we have found counterintuitive examples in our modeling where a longer CS delays the start of execution of a lower-priority ISR until a higher-priority ISR also becomes pending, and hence *avoids* rather than aggravates an AIL violation (for the higher-priority ISR). We believe it may be possible to accidentally construct a practical system where the system may violate its AIL requirements with a shorter CS but may not violate its AIL requirements with a longer CS. This requires further investigation. Assuming that the CS execution time falls into a range circumvents this uncertainty[7].

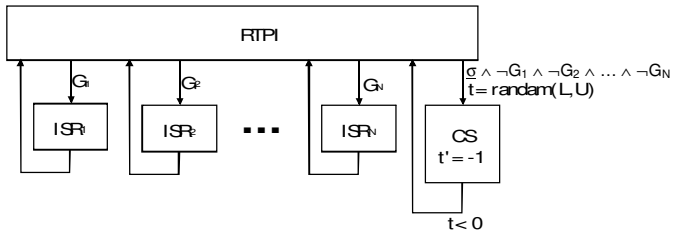

Figure 2: EHM Model Of Software System

IS EHMs signal the SS EHM that an interrupt is asserted through the RSSV belonging to the IS. If the RSSV is set to indicate that the IS requires service, the guard of the transition leaving RTPI will be chosen to allow the SS EHM to enter the appropriate ISR.

In a microprocessor system, the digital hardware assigns priorities. If multiple ISs require service when the SS becomes RTPI, the ISR to be entered is chosen using a fixed priority scheme. This behavior can be captured in the SS EHM by choosing the guards so that a lower-priority ISR cannot be entered while any higher-priority IS requires service.

To illustrate the modeling of priority resolution in hardware, consider a system with three ISs, each with an associated ISR. Assume that IS_1 has the highest priority, followed by IS_2 and then IS_3; and that the semantics of the RSSVs require a value of zero if no service is required, and a value greater than zero if service is required. For this example, suitable guards are shown below.

G_1: $RSSV_1 > 0$
G_2: $RSSV_2 > 0 \wedge RSSV_1 \leq 0$
G_3: $RSSV_3 > 0 \wedge RSSV_2 \leq 0 \wedge RSSV_1 \leq 0$

This strategy for the construction of guards can be extended indefinitely.

[7] Our uncertainty lies with the nature of practical systems rather than with the nature of the mathematical framework. We know it is possible to model such a system within the hybrid system framework, but we are uncertain if any practical system would have this characteristic.

Figure 2 is diagrammatic only, to show the major features of the SS EHM. In Figure 2, the rectangles labeled as ISRs represent the bodies of the ISRs—in a practical SS model, each ISR may entail high complexity and consist of at least several configurations, with multiple transitions leading back to RTPI.

OVERVIEW OF MODELING SOURCES OF INTERRUPT

Figure 3 gives a diagrammatic EHM model of an IS. An IS is modeled to have a unique initial configuration. The configurations of an IS EHM can always be divided into three mutually exhaustive and mutually exclusive categories:

- Configurations in which the IS does not require service.
- Configurations in which the IS requires service.
- Illegal configurations.

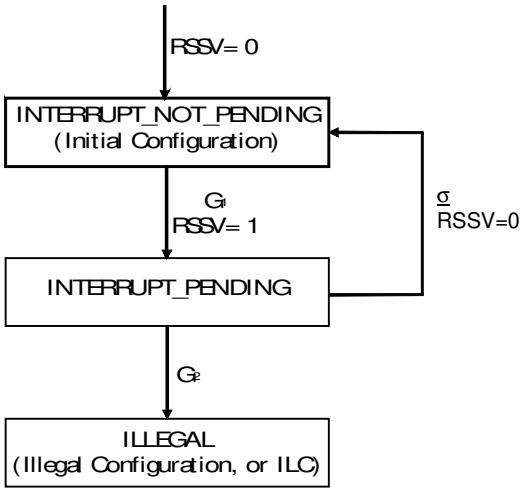

Figure 3: EHM Model Of Interrupt Source

Figure 3 shows the three categories of configurations that will be present an EHM IS model. The microprocessor system will normally reset into a state where no IS is requesting service, hence the initial RSSV of zero. After the system begins operation, at some point the IS may change configuration so that it is requesting service from the SS; G_1 may involve both time and events. Once the IS is requesting service, exactly two outcomes are possible. Either the SS will interact with the IS in a timely fashion to bring the IS to a configuration where it is no longer requesting service (in the figure, $\underline{\sigma}$ represents an OE from the SS EHM which effects a state change in the IS), or else eventually G_2 will be met and the configuration of the IS will be deemed illegal.

Figure 3 is diagrammatic only, in that almost no practical IS can be modeled with only three configurations. Each rectangle in the figure represents a set of configurations. The configuration of the IS will typically evolve through many configurations before the IS requests service, hence G_1 in the figure is not a single guard, but rather a collection of guards on potentially many transitions which define the boundary or the horizon between the set of configurations where the IS is not requesting service and the set of configurations where the IS is requesting service. G_2 and $\underline{\sigma}$ are boundaries or horizons in the same sense.

Although this is not required by the underlying mathematical framework, our design rules (presented later) embrace models where the RSSV is assigned consistently in each transition into a configuration, so that the RSSV is a function of the configuration only.

Each IS model has a single illegal configuration (ILC). All transitions leading into the ILC reflect a situation where the SS was not able to prevent an undesirable state change in the IS through timely interaction. In a practical system, events which lead to the ILC may be a buffer overflow (in the case of a UART), a missed message (in the case of a network communication peripheral), or a periodic interrupt which is not fully processed before the next similar interrupt occurs (a lost "timetick").

The transitions leading into the ILC may represent situations which are very disparate in their nature and severity; the only requirement is that these situations are arbitrarily deemed undesirable in the system. It is not required that each transition leading into the ILC represent a catastrophic event. It would be possible, as a modeling example, to tighten the notion of desirability in order to ensure a system with a certain safety factor or timing margin, or to rule out severe jitter in a timetick ISR.

For the modeling approach presented here, the EHM timing model ceases to represent the modeled system at the ILC boundary. The timing models constructed therefore do not accurately model the behavior of a system in which any ILC is reachable. In practice, this tends to be because the ILC is a modeling "catch-all", which may map to many disparate undesirable configurations in the modeled system. Mathematically, this point is more profound—if no ILC is reachable, then the ILCs can be discarded from the CHM, in effect they are not truly part of the model.

OVERVIEW OF INTERACTIONS BETWEEN SOFTWARE SYSTEM AND SOURCES OF INTERRUPT

As we explored the adequacy of the hybrid system mathematical framework in solving the interrupt latency compatibility problem, about a dozen pairs of EHM models were developed for ISs and their corresponding

ISRs. We have noted similarities in all of our refined models, and these similarities suggest preferred modeling approaches or design rules, which we present here.

1. Each IS EHM maintains a single RSSV, with a value of zero indicating that no service is required, and a value greater than zero indicating that service is required.

2. RSSVs appear in the guards of transitions in the SS EHM leading from the RTPI state, with hardware interrupt priority resolution modeled as discussed earlier.

3. Within each of the ISRs of the SS EHM, assumptions can be made about the configuration of the corresponding IS, so it is known that OEs from the SS will have a specific effect on the configuration of the IS.

4. Each IS has exactly one ILC, which is a modeling "catch-all" for any situation deemed arbitrarily undesirable.

5. Each RSSV is assigned consistently in IS transitions so that it is a function of the IS configuration only.

6. In addition to the RSSV, each IS may maintain additional SVs which allow the ISR model to "sense" the state of the IS and make branch decisions based on this state.

7. An IS is designed so that an RSSV may not go from a value greater than zero to a value of zero or less without intervention from the corresponding ISR[8].

8. Once an ISR is entered, the IS and the ISR may interact in an arbitrary fashion. Typically, the ISR is modeled to be a source of OEs which drive the state of the IS, and the ISR transitions contain guards involving SVs owned by the IS.

9. ISRs each contain at least one exit to the RTPI configuration.

In the following sections, models for typical ISs and their corresponding ISRs are presented. Each of these models adheres to the design rules presented above.

PERIODIC INTERRUPT SOURCES

Figure 4 depicts the EHM model of a periodic IS. In configuration A (the initial configuration), the continuous variable *t* advances with *t'*=1 to mimic elapsed time. When *t* becomes equal to K_1 (the period of the IS), the transition to configuration B is taken, the RSSV is set to signal that the IS requires software intervention, and *t* is set to zero to begin measuring the next period. If the SS interacts with the IS before K_2 elapses by sending $\underline{\sigma}$, the IS drops its request for service. If the SS does not interact with the IS in a timely fashion, K_2 elapses and the IS reaches the ILC. For a typical practical system, $K_2<K_1$, since usually an ISR must begin to run before the next periodic interrupt. Note that K_2 can be set arbitrarily within this constraint to limit jitter of the associated ISR.

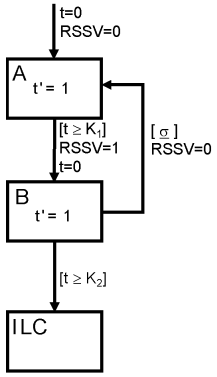

Figure 4: Periodic IS

CONSTANT EXECUTION TIME ISRS

Figure 5 depicts an ISR with constant execution time. This ISR is a typical companion to Figure 4, as a "timetick" ISR often has a constant execution time. On the transition out of RTPI, the OE σ is sent to the IS to effect a state change[9], and an elapsed timer is reset. The SS EHM remains in its single configuration until the modeled constant execution time K is exhausted, then it returns to the RTPI configuration.

[8] Practical ISs never release an asserted interrupt without SS intervention. If this design guideline is violated, guideline (3) cannot be applied, as the ISR has no assurances about the range of configurations of the IS.

[9] This is an application of design rule (3), the ISR can assume that the IS is asserting an interrupt; otherwise, the guard in the SS model would prevent the ISR from being entered.

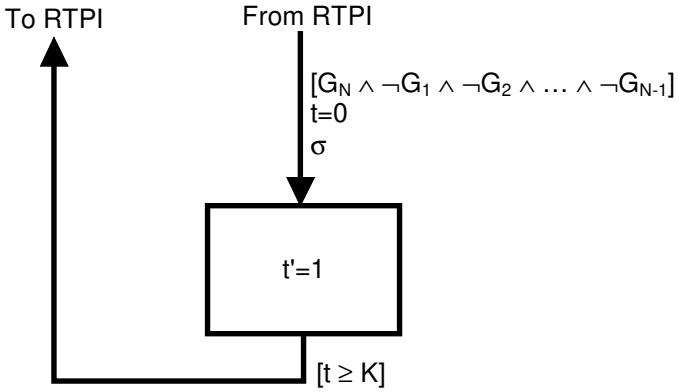

Figure 5: Constant Execution Time ISR

It is easy to see that through the introduction of an additional configuration, a constant execution time ISR can include a startup delay before it interacts with the IS. In practice, an ISR is almost never free to immediately interact with an IS when it begins execution. We hinted previously that the AIL never appears directly in an EHM. It should be clear from the notion of this type of startup delay that both the ISR EHM and the IS EHM need to be considered simultaneously to calculate the AIL from the EHM models.

ALTERNATING EXECUTION TIME ISRS

In practice, it often occurs that an ISR will vary its execution time in a repeating pattern. Such patterns are especially common in "timetick" ISRs which have been "divided down"; that is, implemented so as to perform their primary function during only a rational fraction of their invocations.

Figure 6 depicts an ISR with an alternating execution time. The execution time of this ISR alternates between P_1 and P_2 in an A-B-A-B-… pattern. This ISR is a typical companion to Figure 4. Note that an additional continuous variable *x* is used to allow the ISR to hold state between modeled invocations. It is easy to see that this mechanism can be extended to arbitrary repeating patterns.

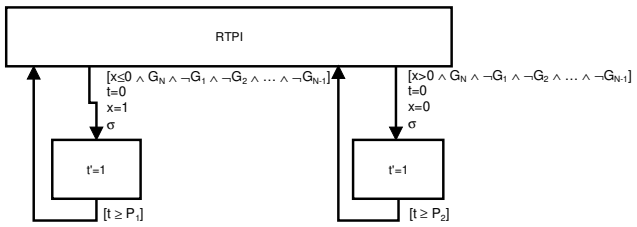

Figure 6: Alternating Execution Time ISR

QUEUED RECEIVER PERIPHERAL INTERRUPT SOURCE AND ISR

In practical systems, primarily due to queued ISs, an ISR may have an execution time that increases with increasing interrupt latency. Phrased more colloquially, the longer you wait, the worse it can get. We feel that any modeling framework must capture this aspect of a practical system. An example of a typical queued communication peripheral is the 16550 UART, which contains a queue of 16 bytes.

We are aware of two modeling approaches that can treat such devices within the framework of hybrid system EHM models.

1. The queued IS can be modeled as having a continuous queue state (although a practical IS will have a discrete queue state). This is the approach presented here.

2. The queued IS can be modeled as having a discrete queue state, i.e. the queue state can be captured in the EHM configuration. For an IS containing a small queue, this approach is practical. We will not supply the details here, as this approach is intuitive.

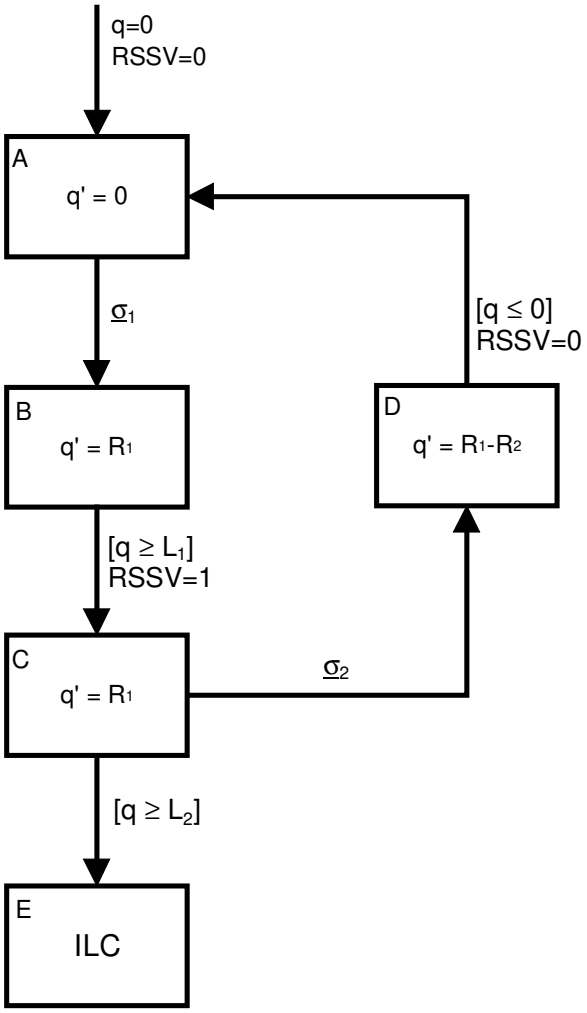

Figure 7: EHM Model Of Queued Receiver IS

We prefer modeling the queue state as continuous, as it reduces the number of configurations in the resulting CHM. In practical systems, an ISR tends to loop until the IS queue is depleted, and the execution time cost per item removed is approximately constant. We feel fairly confident in modeling a queue with a discrete state using a continuous variable, in that the continuous model of the ISR can be designed to require at least as much execution time as the discrete model, in the same sense as $\forall x, \lfloor x \rfloor \leq x$. However, we do not claim that this replacement of a discrete queue state by a continuous queue state is mathematically valid or without potential peril[10].

[10] The implicit (and invalid) assumption behind such a substitution is that a longer ISR execution time is "worse", so that a system which shows AIL compatibility with the continuous IS model will also show AIL compatibility with the discrete IS model. Although we believe this logical implication will hold for almost any practical system, it will not hold in the general case, and this can be shown by a compact counterexample.

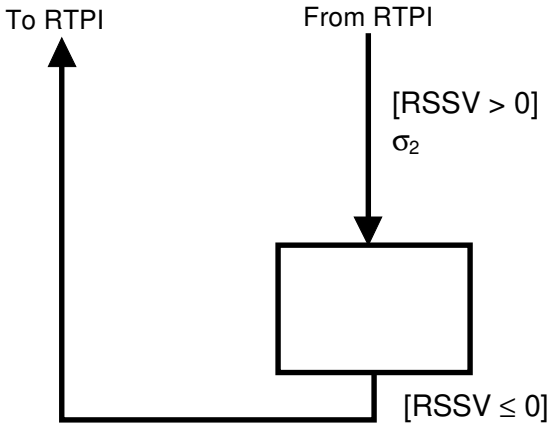

Figure 8: EHM Model Of Queued Receiver ISR

Figures 7 and 8 show EHM models of a queued receiver IS and the associated ISR, respectively. In interpreting the figures, assume that *q* represents the occupancy of the queue, that the IS will assert a hardware interrupt when the queue fills to L_1, that the queue has capacity L_2, that the queue can fill at the rate R_1, and that the software can empty the queue at rate R_2.

The IS EHM (Figure 7) begins in configuration A with q=0 (an empty queue) and RSSV=0 (not requesting CPU attention). At an arbitrary time, environmental input event $\underline{\sigma}_1$ will cause the IS to assume configuration B. In configuration B, the queue begins to fill at rate R_1. When the queue reaches L_1, the IS assumes configuration C, in which it is requesting attention from the CPU. If the software ISR fails to intervene in time and the queue reaches L_2, the ILC will be entered; otherwise the ISR will send event $\underline{\sigma}_2$, which will cause the queue to empty at the rate R_1-R_2 until the queue is empty. The ISR EHM model (Figure 8) is substantially simpler, it simply waits in a configuration for the IS queue to empty.

NON-ATOMIC ISRS

By *atomic ISR* we mean an ISR which will always run to completion because it cannot be interrupted by any other ISR. We have found that software developers intuitively avoid non-atomic ISRs. Most experienced software developers have seen more than one elusive software defect traced to improper treatment of interrupts, and consequently will intuitively attempt to limit the freedom of the SS by allowing only atomic ISRs. Most microprocessor SSs involve only atomic ISRs.

It occasionally occurs, however, that a collection of ISs and ISRs are mutually incompatible with respect to AIL, and that compatibility can be achieved by making at least one ISR interruptible by higher-priority ISRs. Fortunately, most microprocessor interrupt hardware is very flexible, in that substantial control is available over when a certain priority of ISR can be interrupted, and by which other ISRs it can be interrupted.

If reachability analysis shows that AIL violations are possible, we propose the following modeling and software design approach to create a system without potential AIL violations.

A lower-priority ISR should be divided into an integral number of segments of approximately equal execution time. The number of segments must be chosen so as to relieve the AIL problem and create a system where no AIL violation is possible, i.e. for a system that is almost satisfactory, N=2 (a bisection) would be appropriate, but for more severe timing problems a larger N may be required. Figure 9 shows the SS model resulting from our approach (with N=3, a trisection) applied to a system with two ISRs, where ISR_1 is the higher-priority ISR and ISR_2 is the lower-priority ISR. Note that the model of ISR_1 now appears in three places in the SS EHM. Note also that the guards implement priority resolution, and only allow ISR_2 to be interrupted at certain points.

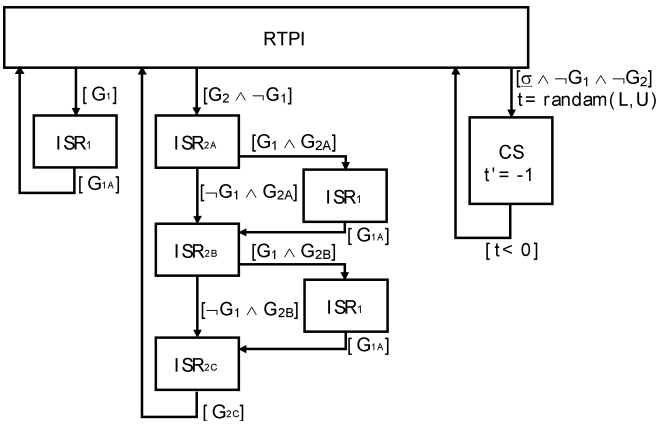

Figure 9: EHM Model Of SS With Non-Atomic ISR

If reachability analysis shows that the modified system cannot exhibit AIL violations, then the actual software should be modified by inserting interrupt windows in the lower-priority ISR. Such windows would typically consist of an instruction which enables interrupts immediately followed by an instruction which disables interrupts. These windows would allow the lower-priority ISR to be interrupted, but only at certain points. We feel that an approach using interrupt windows is safer than allowing the lower-priority ISR to be interrupted at an arbitrary point.

ADDITIONAL MODELING SCENARIOS

The examples presented in the preceding sections illustrate the hybrid system modeling of ISs and the SS. The examples have been chosen to demonstrate the EHM modeling of essential features of a microprocessor system.

In practice, ISs and ISRs are not qualitatively different than the examples, but they are often far more complex. To model complex ISs, it has sometimes been useful to create an IS which consists of multiple EHMs. For example, to model a network communication peripheral which has a receive queue consisting of RAM storage for a certain integral number of messages, it might be natural to decompose the model into a transmit EHM, a receive EHM, a receive queue EHM, and an EHM which generates network physical medium timing. These EHMs interact using the two interface mechanisms allowed under the hybrid system framework (SVs and events which are generated by one EHM and accepted by another). It has also been convenient to use guards where the atomic formulas involve comparisons of SVs against each other rather than against constants. Collusion between ISs can also be modeled using SVs and passed events.

We have not yet encountered and do not anticipate encountering a practical system which cannot be evaluated for interrupt latency compatibility using the hybrid system method we have proposed.

VERIFICATION ALGORITHM

Since an algorithm for verifying that the interrupt latency compatibility requirement is satisfied is equivalent to an algorithm for checking reachability in the CHM, we present a reachability algorithm in this section.

The essence of our algorithm is as follows. We start from an illegal configuration q'. It will be reached from a legal configuration q only when one IS goes without service from the SS for too long; that is, when the AIL of the IS is violated or equivalently when G=[t>b] becomes true. This is illustrated in Figure 10. Such a transition is called illegal. Denote the set of illegal dynamic transitions from q by $DT(q, Q_b)$.

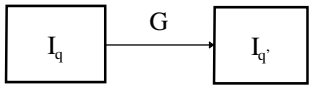

Figure 10. From Legal to Illegal

In order to guarantee that a transition $(q,G,q') \in DT(q,Q_b)$ will not take place and result in the violation of an AIL requirement, some other transitions from q (to

some legal configurations) must occur before (q,G,q') can occur. In other words, (q,G,q') must be preempted by some other transitions. This may or may not be possible depending on the continuous variable x at q.

To find a preemptive condition for transition (q,G,q'), we recall our assumption that the invariant I_q is violated if and only if one of the guards at q is satisfied. Under this assumption, the transition (q,G,q') will be preempted by another dynamic transition from q if and only if the time when G will become true is larger (that is, later) than the time when will I_q become false. Denote the time when the predicate P will become true by T(P). We can calculate T(P) as follows, starting from atomic formulas.

Case 1: $P=[s_i \geq C_i]$ or $P=[s_i > C_i]$, where s_i is an SV and C_i is a constant. By our assumption on continuous dynamics, s_i has a constant rate: $s_i = r_i$. Therefore, T(P) can be calculated as follows.

$$T_{\min}(P) = \begin{cases} 0 & if \quad s_i > C_i \\ (C_i - s_i)/r_i & if \quad s_i < C_i \wedge r_i > 0 \\ \infty & otherwise \end{cases}$$

Case 2: $P=[s_i \leq C_i]$ or $P=[s_i < C_i]$.

$$T_{\min}(P) = \begin{cases} 0 & if \quad s_i < C_i \\ (C_i - s_i)/r_i & if \quad s_i > C_i \wedge r_i < 0 \\ \infty & otherwise \end{cases}$$

Case 3: $P=P_1 \wedge P_2$.

$T(P)=\max\{T(P_1), T(P_2)\}$.

Case 4: $P=P_1 \vee P_2$.

$T(P)=\min\{T(P_1), T(P_2)\}$.

Case 5: P=true.

$T(P)=0$.

Case 6: P=false.

$T(P)=\infty$.

Now, the condition to preempt an illegal transition $(q,G,q') \in DT(q,Q_b)$ is given by

$$pc(q,G,q')=[T(G)>T(\neg I_q)].$$

Here we assume that the weakest precondition is always satisfied, that is, $G \Rightarrow wp(q,G,q')$. Otherwise, we will modify the preemptive condition as

$$pc(q,G,q')=[T(G \wedge wp(q,G,q'))>T(\neg I_q)].$$

Using this preemptive condition, we can split the configuration q into two sub-configurations: good configuration q_g and bad configuration q_b by partitioning the invariant I_q as

$$I_{qg}=I_q \wedge pc(q,G,q')$$
$$I_{qb}=I_q \wedge \neg pc(q,G,q').$$

Clearly, the dynamics of and the transitions leaving and entering the configurations q_g and q_b are the same for q, except that the transition (q,G,q') is now impossible.

If there is more than one illegal dynamic transition at q, then we will split q into q_b and I_q as follows.

$$I_{qg}=I_q \wedge (\wedge_{(q,G,q) \in DT(q,Qb)} pc(q,G,q'))$$
$$I_{qb}=I_q \wedge \neg(\wedge_{(q,G,q) \in DT(q,Qb)} pc(q,G,q')).$$

From the above discussion, we can now formally describe our reachability algorithm.

REACHABILITY ALGORITHM

Input

- The model of the system

  $CHM=(Q,\Sigma, D, I, E, (q_0, x_0))$.

- The set of illegal configuration

  $Q_b \subseteq Q$.

Output

- The set of good invariants I_{qg}

  $GI=\{ I_{qg}: q \in Q\}$.

Execution

Initialization

1. $P:=\{ q_g: q \in Q\}$;
2. $R:=\{ q_b: q \in Q\}$;
3. For all $q \in Q-Q_b$ do

l_{qg}:=l_q;

l_{qb}:=false;

4. For all q∈Q_b do

   l_{qg}:=false;

   l_{qb}:= l_q;

5. Stop:=true;

Iteration

6. For all q such that l_{qg}≠false do

   H:=∧(q,G,q')∈ DT(q,R) pc(q,G,q');

   If l_{qg}≠l_q∧H then

   l_{qg}:=l_q∧H;

   l_{qb}:=l_q∧¬H;

   Stop:=false;

7. If Stop=true, then go to 10;

8. Stop:=true;

9. Go to 6;

10. End.

This algorithm takes the CHM model of the system and the set of illegal configurations Q_b as inputs. Both CHM and Q_b can be obtained automatically from the EHMs as described in the previous section. The algorithm first makes two copies q_g and q_b for each configuration q. It then repeatedly partitions the invariant I_q into *good* invariant I_{qg} and *bad* invariant I_{qb}. The initial partition is straightforward. Subsequent partitions require the computation of preemptive conditions. The output of the algorithm is the set of good invariants GI={ I_{qg}: q∈Q}.

Various information can be derived from GI={I_{qg}: q∈Q}. For example, interrupt latency requirements will not be violated if and only if for the given intial condition (q_0, x_0), the good invariant at the initial configuration I_{q0g} is satisfied by x_0, that is,

$I_{q0g}|_{x=x0}$=true.

EXAMPLE: HYBRID SYSTEM MODEL OF MICROPROCESSOR SYSTEM WITH REACHABILITY SOLUTION

To illustrate our approach, we consider a small example. The system consists of two sources of interrupts, IS_1 and IS_2. Both request interrupt service periodically, with periods T_1 and T_2, respectively. The corresponding interrupt service routines ISR_1 and ISR_2 are executed on the same CPU, with priority given to IS_1. It takes the CPU C_1 and C_2 units of time to execute ISR_1 and ISR_2 respectively. We assume that both ISRs are atomic. The interrupt latencies cannot exceed the deadlines $d_1=T_1-C_1$ and $d_2=T_2-C_2$ respectively. In other words, the interrupt service routines must be completed before the next period. We will show how this problem can be solved using our approach.

To model the interrupt sources IS_i, i=1,2, we use the EHM_i shown in Figure 11.

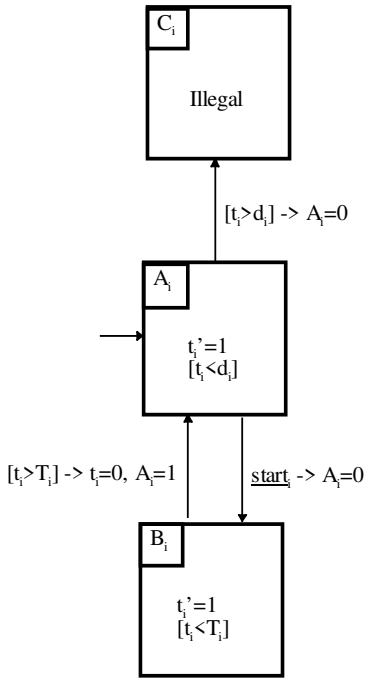

Figure 11. ISR_i EHM

EHM_i has three vertices: in configuration A_i an interrupt is pending, and configuration C_i is illegal. SV t_i models the clock and SV A_i indicates whether the EHM_i is in vertex A_i (it is the RSSV). Constant $d_i=T_i-C_i$ is the deadline for waiting. Event $start_i$ represents the start of the interrupt service routine ISR_i. The initial vertex is A_i and is indicated by arrow ->.

The ISRs are modeled by EHM_3, shown in Figure 12.

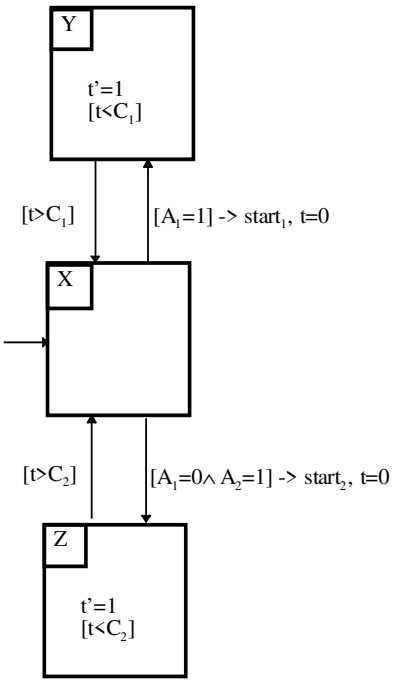

Figure 12. EHM$_3$

EHM$_3$, representing the SS, has three vertices as follows.

X: ready to execute interrupt service routines;
Y: executing ISR$_1$; and
Z: executing ISR$_2$.

The parallel composition EHM$_1$||EHM$_2$ is shown in Figure 13, where only four configurations A_1A_2, A_1B_2, B_1A_2, and B_1B_2 are legal. In the figure, we do not show the transitions between illegal configurations because they are not important. In fact, all the illegal configurations can be combined into one.

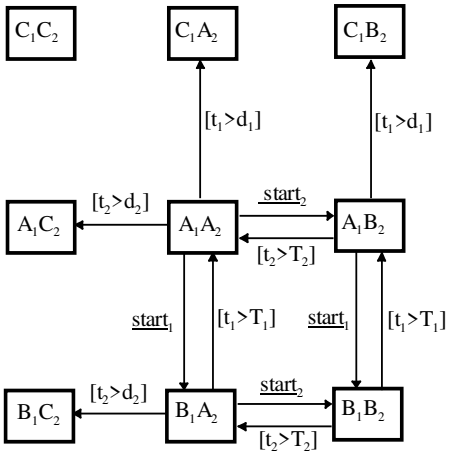

Figure 13. EHM$_i$ ||EHM$_2$

The overall system is modeled by

$$CHM=EHM_1||EHM_2||EHM_3$$

which can be constructed similarly.

Using our algorithm, we can determine if the illegal configurations in the CHM are reachable from the initial configuration. This, of course, depends on the values of T_i and C_i. The following cases have been verified.

Case 1: T_1=5, T_2=4, C_1=3, C_2=2. In this case, we find that the illegal configurations are reachable from the initial configuration. Therefore, the interrupt latency requirement is violated.

Case 2: T_1=8, T_2=3, C_1=5, C_2=2. In this case, we find that the illegal configurations are reachable from the initial configuration. Therefore, the interrupt latency requirement is violated.

Case 3: T_1=5, T_2=8, C_1=1, C_2=1. In this case, we find that the illegal configurations are not reachable from the initial configuration. Therefore, the interrupt latency requirement is not violated.

Case 4: T_1=17, T_2=4, C_1=3, C_2=1. In this case, we find that the illegal configurations are reachable from the initial configuration. Therefore, the interrupt latency requirement is violated.

Case 5: T_1=5, T_2=6, C_1=3, C_2=2. In this case, we find that the illegal configurations are reachable from the initial configuration. Therefore, the interrupt latency requirement is violated.

Case 6: T_1=80, T_2=40, C_1=3, C_2=2. In this case, we find that the illegal configurations are not reachable from the

initial configuration. Therefore, the interrupt latency requirement is not violated.

SOFTWARE IMPLEMENTATION OF ALGORITHM

For a practical microprocessor system, determining interrupt latency compatibility using the hybrid system algorithm presented here requires a computer solution. The upper bound on the number of CHM configurations for which a manual determination of reachability is practical is on the order of 20. In contrast, the practical microprocessor systems we have encountered would generate CHMs with on the order of 40,000 to 100,000 configurations. Reachability algorithms cannot be applied to such systems by hand.

A fair number of problems can be embedded in the hybrid system framework, including a classic steam boiler problem [2]. It has been our experience that most problems which are amenable to modeling using a hybrid system framework involve either software alone or software in control of a physical plant. We feel that the number of problems which require analysis using a hybrid system framework is adequate to spawn the commercial development of computer tools. We also feel that the number of such problems is growing, due to the proliferation of computer-controlled systems. As of this writing, we are aware of three companies engaged in the for-profit development of computer tools for the analysis of hybrid systems.

ACKNOWLEDGMENTS

We would like to gratefully acknowledge the support of the management team at Visteon (Dan Presidio, Dave Patterson, Chuck Minear, Dave Avery, Marian Mahoney, Joe DeVoe, and Nick Sarafopoulos), whose quest for product excellence led to the funding of this effort and the freedom to pursue a solution to this problem in the workplace. We also gratefully acknowledge the support of the technical professionals at Visteon who shared with us their expertise and exhaustive insight into this problem and the constraints on the solution; especially Joao Silva, Kevin Tiedje, Bob Crawford, and Karl Overberg.

CONTACT

For further information, please contact Dr. Feng Lin, Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, Tel: (313) 577-3428, Fax: (313) 577-1101, email: flin@ece.eng.wayne.edu, home page: www.ece.eng.wayne.edu/~flin.

REFERENCES

1. Audsley, Neil C.; Burns, Alan; Davis, Robert I.; Tindell, Ken W.; Wellings, Andy J.. Fixed Priority Pre-emptive Scheduling: An Historical Perspective. Real-Time *Systems v* 8 n 2-3 Mar-May 1995 pp. 173-198.
2. M. Heymann, F. Lin and G. Meyer, 1997. Control Synthesis For A Class Of Hybrid Systems Subject To Configuration-Based Safety Constraints. NASA Technical Memorandum *112196.*
3. Park, Dong-Won; Natarajan, Swaminathan; Kanevsky, Arkady. Fixed-priority Scheduling Of Real-time Systems Using Utilization Bounds. Journal *of Systems and* Software v 33 n 1 Apr 1996 pp. 57-63.
4. Park, Dong-Won; Natarajan, S.; Kanevsky, Arkady; Kim, Myung Jun. Generalized Utilization Bound Test For Fixed-Priority Real-Time Scheduling Proceedings *of the 1995* 2nd International *Workshop* on Real-Time Computing *Systems and Applications* Oct 25-27 1995 Tokyo Japan pp. 73-77.
5. Liu, C.L.; Layland, James W. Scheduling Algorithms for Multiprogramming In A Hard Real-Time Environment. Journal Of The ACM, v 20 n 1, January 1973, pp. 46-61.
6. M. Heymann, F. Lin, and G. Meyer, “Synthesis And Viability Of Minimally Interventive Legal Controllers For Hybrid Systems”, Discrete Event Dynamic Systems: Theory and Applications, 8(2), pp. 105-135.

DEFINITIONS/ACRONYMS/ABBREVIATIONS

AIL
: Allowable interrupt latency, the maximum delay (without ill effects) between the assertion of an interrupt in hardware and the start of execution of the associated interrupt service routine.

CHM
: Composite hybrid machine, a collection of configurations and transitions formed by performing a parallel composition of several EHMs. Note that a CHM is structurally identical to an EHM (form of configurations and transitions), but a CHM will contain many more configurations than an EHM.

Configuration
: The discrete part of the state vector of an EHM or CHM. In our drawings, configurations are shown as rectangles and are linked together by transitions. Because an EHM or CHM also contains continuous state variables, the state vector of an EHM or CHM is hybrid. We use configuration to refer to the discrete part only.

CS
: Critical section, a sequence of machine instructions executed with hardware interrupts disabled. Critical

sections are commonly used to properly share resources (data structures, hardware) between ISR and non-ISR software components, or to guarantee reliable timing of a sequence of instructions. Critical sections are relevant to the interrupt latency compatibility problem because they may delay the start of an ISR.

EHM
: Elementary hybrid machine, a collection of configurations, dynamics, and transitions.

ILC
: Illegal configuration, a configuration in an EHM which is deemed arbitrarily undesirable.

IS
: Interrupt source, a hardware peripheral which can generate a hardware interrupt, such as a timer or a communication peripheral.

ISR
: Interrupt service routine, a software component which is executed in response to a hardware interrupt.

OE
: Output event, an event generated by an EHM. An output event of one EHM may be an input of another EHM.

RMA
: Rate-monotonic analysis, a body of mathematical results derived from a paradigm involving software processes with a constant rate of execution and a constant execution time.

RSSV
: Requesting service shared variable, a shared continuous variable used by an interrupt source EHM to indicate to the SS EHM that service is requested.

RTPI
: Ready to process interrupts, a single configuration in our model of a software system, during which the software is free to process interrupts by executing an ISR.

SS
: Software system, the EHM model of the software system including ISRs and CSs.

SV
: Shared variable, a continuous variable which is assigned by exactly one EHM and may be tested by an unlimited number of other EHMs.

UART
: Universal asynchronous receiver transmitter, a common communication peripheral which can receive and send serial characters.