A Framework for Interactive Analysis of Timing Constraints in Embedded Systems

Rajesh K. Gupta
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue
Urbana, Illinois 61801.

Abstract

An important goal of embedded system co-synthesis is to realize system designs under constraints on timing performance. We present applicable constraints and the notion of satisfiability *of a given set of constraints. We describe a two-level system model that is useful for carrying out constraint analysis in presence of the timing and execution uncertainty inherent in embedded systems. We conclude by presenting a framework to determine constraint satisfiability and to interactively debug constraint violations. Examples are presented to show the utility of our approach.*

1 Introduction

Timing relationships in embedded systems originate from their requirement to maintain timely interactions with a reactive environment. Examples of such systems can be found in medical instrumentation, process control, automated vehicle control, and networking systems. Several approaches to system design and CAD treat an embedded system implementation as consisting of interacting hardware and software components. An overview of these approaches in the general area of hardware-software co-design is presented in [1]. These embedded *computing* systems typically consist of several concurrent and interacting modules that often operate with a wide disparity in the rate of execution. Thus, an analysis of constraints on the timing relationships is crucial to a systematic design methodology. We consider timing constraints as bounds on operation delays, or on the interval between operation start times, or as bounds on the execution rate of periodic operations. Constraints may also be specified relative to a specific context or control-flow in the system specification. Constructs for specifying these constraints are presented later.

The chief contribution of this paper is a two-level system model and a framework for analysis of timing constraints. This analysis is useful to the embedded system designer in identifying performance bottlenecks in system design.

This paper is organized as follows. In Section 2 we introduce our system model. We provide an overview of the tasks in performance analysis for embedded system in Section 3 followed by a summary of the tests to ensure constraint satisfiability in Section 4. The description of individual algorithms is beyond the scope of this paper and appropriate references have been provided. We present a framework for specifying and checking constraint satisfiability in Section 5. Section 6 summarizes results and contributions.

2 A Two-Level System Model

A co-synthesis approach to embedded computing system (ECS) is based upon compilation and synthesis techniques for digital hardware. Therefore, it is required that the input language used for system specification has a synthesis path to hardware realization. We use a C-based Hardware Description Language(HDL), called HardwareC [2] though other HDLs, such as VHDL or Verilog, can be used as well.

The basic entity for specifying system functionality is a *process*. A process executes concurrently with other processes in the specification. A process restarts itself on completion of the last operation in the process body.

The HDL description is compiled into a set of graph models. A graph model provides an abstraction between objects modeled as vertices and a relation between the objects, modeled as edges or hyper-edges. When object vertices represent operations or other executable entities, a typical execution semantics associated with the graph model is to enable a vertex for execution once its predecessor has completed execution. In case of multiple predecessors, a choice needs to be made in enabling the vertex for execution once *all* or *any* of its predecessors have completed execution. Graph models that

0-8186-7243-9/96 $05.00

choose one of the two interpretations are referred to as *unilogic* graphs [3]. In some graph models, both semantics are allowed. Such graphs are called *bilogic* [4]. Our system model consists of unilogic as well as bilogic graphs at the process and operation levels of abstraction as described below.

1. Process-level consists of a unilogic process graph that models processes and their synchronization interactions. An embedded system is modeled as a process graph, $G_P(V_P, E_P)$. A vertex $p_i \in V_P$ represents a process with an independent thread of control. We assume that all processes are concurrently active. The process interaction is by means of *enable signals* that enable the execution of other processes. An enable signal from process p_i to process p_j is represented by a directed edge $(p_i, p_j) \in E_P$. Each edge (p_i, p_j) has a delay $d_P(p_i, p_j)$ associated with it which represents the delay in invocation of p_j after start of p_i. A new *instance* of process, p_i, starts executing after *all* its predecessor processes have issued enable signals for p_i. This execution semantics allows for multiple instances of a process to execute simultaneously.

Given a set of processes, modeled as nodes in a process graph, two types of inter-process relations are modeled:

- *Sequencing* is indicated by an edge between two processes
- *Concurrency* is indicated by the forking of multiple edges from a process.

Use of only two relations between processes makes it possible to represent a process-level model algebraically, upon which the rate analysis is based [5]. However, at the process-level, conditional invocation of processes is not allowed. This limitation is not overly restrictive, since a conditional selection between two process graphs can be easily modeled at the operation-level that is described next.

2. Operation-level consists of a bilogic flow graph that models operations and their dependencies. A flow graph captures both data and control dependencies in a hierarchical directed acyclic polar graph. Each process, p_i, in the process graph is modeled as an operation-level flow graph, $G_i(V_i, E_i, \chi)$. The vertex set $V_i(G)$ represents *language-level* operations and special *link* vertices that are used to encapsulate hierarchy. A link vertex induces (single or multiple) calls to another flow graph which may represent a shared resource or body of a loop operation. The edge set $E_i(G)$ represents dependencies between operation vertices.

Function χ associates a Boolean (enabling) expression with every edge. In the case of edges incident from a condition vertex or incident to a join vertex, the enabling expression refers to the condition under which the

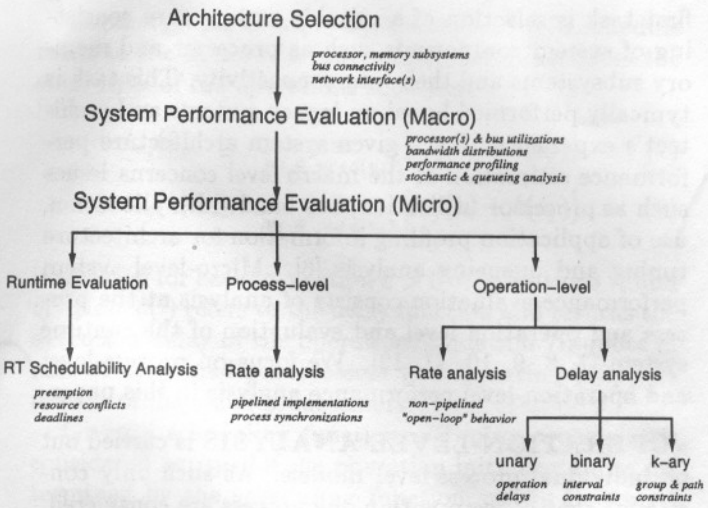

Figure 1: Performance analysis for embedded systems.

successor node for the edge is enabled. These expressions determine the flow of control through the graph model which includes both concurrency as well as conditional selection.

Given a set of operations, modeled as vertices in a flow graph, three types of inter-operation relations are modeled:

- *Sequencing* is indicated by an edge between two operations
- *Concurrency* is indicated by *conjoined* forking of multiple edges from an operation
- *Conditional choice* is indicated by a *disjoined* forking of multiple edges from an operation.

Finally, we note that the loop link and wait operations introduce uncertainty over the precise delay and order in which various operations are invoked. Due to this uncertainty, a system model containing these operations is called a *non-deterministic* model and operations with variable delay are termed as non-deterministic delay, or ND, operations. Apart from the ND operations, we assume that we are given estimates of the execution time of the operations in the flow graph for all processes. These estimates may be crude because execution times may be value dependent and may also be execution-context sensitive.

3 ECS Performance Evaluation

Figure 1 shows an overview of the performance analysis tasks for embedded systems. These tasks are strongly linked to the level of abstraction in the design process. For a given embedded application, the

first task is selection of a suitable architecture consisting of system components such as processor and memory subsystems and their interconnectivity. This task is typically performed based on legacy systems and architect's experience. For a given system architecture performance estimation at the macro level concerns issues such as processor utilization, bus bandwidth allocation, use of application profiling information for architecture tuning and queueing analysis [6]. Micro-level system performance evaluation consists of analysis at the process and operation level and evaluation of the runtime system [7, 8, 9, 10, 11, 12]. We focus on process-level and operation-level performance analysis in this paper.

• **OPERATION-LEVEL ANALYSIS** is carried out on individual process-level models. As such only constraints that concern within one process are considered. In addition, inter-process synchronization is not considered at this level of analysis. Operation-level analysis consists of delay and rate analyses.

◦ **Delay Analysis** refers to analysis of timing constraints on execution delay of individual or groups of operations. In particular, we consider binary constraints that define (upper and lower) bounds on time-intervals of operation pairs. In general, there are thirteen possible interval relationships as indicated in Table 1. Here symbols t_a and d_a represent execution start time and execution delay of operation a. Our input language provides for specification of all possible timing interval relationships which are subsequently captured into a group of minimum and maximum (binary) delay constraints. A minimum (binary) delay constraint, $l_{ij} \geq 0$ from operation vertex v_i to v_j is defined as:

$$t_{v_j}(k) \geq t_{v_i}(k) + l_{ij} \tag{1}$$

Similarly, a maximum delay constraint, $u_{ij} \geq 0$ is defined as:

$$t_{v_j}(k) \geq t_{v_i}(k) + u_{ij} \tag{2}$$

◦ **Rate Analysis** concerns the interval of time between successive executions of an operation. In particular, rate constraint on an input(output) operation refers to the rates at which the data is required to be consumed(produced). We assume that each execution of an input(output) operation consumes(produces) a *sample* of data. Assuming a synchronous execution model with cycle time τ, we define the rate of execution at invocation k of an operation v_i as the inverse of the time interval between its current and previous execution. That is,

$$r_i(k) \triangleq \frac{\tau}{t_{v_i}(k) - t_{v_i}(k-1)} \ (\text{cycle}^{-1}) \tag{3}$$

By convention, the instantaneous rate of execution is 0 at the first execution of an operation ($t_0 \rightarrow -\infty$).

Constraint type	*Description*
A before B	$t_a + d_a < t_b$
A meets B	$t_a + d_a = t_b$
A overlaps B	$t_b - t_a < d_a$
A finishes by B	$t_a + d_a = t_b + d_b$
A during B	$t_a > t_b$ and $t_a + d_a < t_b + d_b$
A finishes B	$t_a > t_b$ and $t_a + d_a = t_b + d_b$
A is overlapped by B	$t_a - t_b < d_b$
A is met by B	$t_a + d_a = t_b$
A after B	$t_b + d_b < t_a$
A contains B	$t_a < t_b$ and $d_a > d_b$
A starts B	$t_a = t_b$ and $d_a < d_b$
A equals B	$t_a = t_b$ and $d_a = d_b$
A is started by B	$t_a = t_b$ and $d_a > d_b$

Table 1: Possible binary timing constraints.

For a flow graph, G, its *rate of reaction*, is defined as the rate of of execution of its source operation, that is, $\varrho_G(k) \triangleq r_0(k)$. The reaction rate is a property of the graph model and it is used to capture the effect on the runtime system and the type of implementation chosen for the graph model. To be specific, the choice of a non-pipelined implementation of G leads to

$$\varrho_G(k)^{-1} = \lambda_G(k) + \gamma_G(k) \tag{4}$$

where $\gamma(k)$ refers to the *overhead delay*, that represents the delay in re-invocation of G. $\gamma(k)$ may be a fixed delay representing the overhead of a runtime scheduler or it may be a variable quantity representing delay in case of conditional invocation of G.

The actual execution delay or the latency, $\lambda_G(k)$, refers to the delay of the longest path in G. This path may contain ND operations in which case the latency is variable and not bounded. We define the time interval $[t_{v_0(G)}(k+1) - t_{v_N(G)}(k)]$ as the *overhead* $\gamma_G(k)$ where v_0 and v_N refer to the source and sink operations respectively. If G is not a root-level flow graph, then there exists a parent flow graph G_+ that calls G using a link operation, say v. The overhead time for G then refers to the time-interval in successive invocations of v in G_+.

A *minimum rate constraint*, r_l (cycles^{-1}), on an input/output operation defines the lower bound on the execution rate of operation v_i. Similarly, a *maximum rate constraint*, r_u (cycles^{-1}), on an I/O operation defines the upper bound on the execution rate of operation v_i. That is,

$$\tau \cdot {r_u}^{-1} \leq t_{v_i}(k) - t_{v_i}(k-1) \leq \tau \cdot {r_l}^{-1} \tag{5}$$

In general, when considering rate of execution of v_i we must consider the successive executions of v_i that may belong to separate invocations of G. A *relative rate constraint* on an operation, v_i, with respect to a graph

model, G, is a constraint on the rate of execution of v_i for the same invocation of G. The relative rate of execution expresses rate constraints that are applicable to a specific *context* of execution as expressed by the control flow in G. The problem of operation delay constraint analysis is then to determine a feasible solution to Inequalities 1,2 and 5 for a given set of hierarchical flow graphs, G.

• **PROCESS-LEVEL ANALYSIS** is carried out on the process graph models that include inter-process synchronization operations. The marginal rate of execution of a process is defined as the number of executions of the process per unit of time. Let $t_{p_i}(k)$ denote the time when the kth execution of process p_i starts. Since the time between successive executions of a process is not constant, we define the *average execution rate* of a process p_i to be

$$\overline{r} = \left[\lim_{n \to \infty} \frac{\sum_{k=0}^{n-1} t_{p_i}(k+1) - t_{p_i}(k)}{n} \right]^{-1} \quad (6)$$

if the above limit exists. It can be shown that for a finite process graph with finite edge delays the above limit exists and can be efficiently computed [5]. The problem of rate analysis is to find the upper and lower bounds on the average rate of execution of each process in a given process graph model of an ECS.

Thus, the rate analysis finds an interval $[r_l(p_i), r_u(p_i)]$ for each process p_i such that the average rate of execution of the process is guaranteed to lie in this interval. This information about the average rate of execution of a process is used to compute the bounds on the average execution rates of operations in the flow graph. We now provide a summary of the results to solve constraint analysis problems.

4 Constraint Satisfiability Tests

At the operation-level constraint satisfiability is related to existence of a feasible schedule of operations. For each invocation of a flow graph, an operation is invoked zero, one or many times depending upon its position on the hierarchy of the flow graph model. The execution times $t_v(k)$ of an operation v are determined by two separate mechanisms: (a) The runtime scheduler, Υ, and (b) The operation scheduler, Ω. The runtime scheduler determines the invocation times of flow graphs, which may be as simple as fixed-ordered where the selection is made by a predefined order (most likely by the system control flow). This is typically the case in hardware implementations where the graph invocation is purely a subject of system control flow. On the other hand, software implementations of the runtime scheduler are based on the choice of the runtime environment.

Given a graph, $G = (V, E)$, the selection of a schedule refers to the choice of a function, Ω that determines the start time of the operations such that

$$t_{v_i}(k) \geq \max_{v_j \in pred(v_i)} \left[t_{v_j}(k) + \delta(v_j) \right] \quad (7)$$

$$t_{v_i}(k) \geq \min_{v_j \in pred(v_i)} \left[t_{v_j}(k) + \delta(v_j) \right] \quad (8)$$

is satisfied for each invocation $k > 0$ of operations v_i and v_j. Here $\delta(\cdot)$ refers to the delay function and returns the execution delay of the operation. Equation 7 applies in case of conjoined predecessors and Equation 8 applies in case of disjoined predecessors.

Given a scheduling function, a timing constraint is considered *satisfied* if the operation initiation times determined by the scheduling function satisfy the corresponding constraint inequalities (1, 2 and 5). Clearly, the satisfaction of timing constraints is related to the choice of the schedulers Ω and Υ. In general, choice of a particular operation scheduling mechanism depends upon the types of operations supported and the resulting control hardware or software required to implement the scheduler.

For constraint analysis purposes, it is not necessary to determine a schedule of operations, but only to verify the *existence* of a schedule. Since there can be many possible schedules, constraint satisfiability analysis proceeds by identifying conditions under which no solutions are possible. A timing constraint is considered *inconsistent* if it can not be satisfied by *any* implementation of the flow graph model. Since the consistency of constraints is independent of the implementation, these are related to the structure of the graphs.

The operation-level delay constraints are abstracted in an edge-weighted *constraint graph model*, $G_T(V, E_f \cup E_b, \Delta)$, where the edge set contains forward edges E_f representing minimum delay constraints and backward edges E_b representing maximum delay constraints. An edge with weight $\delta_{ij} \in \Delta$ on edge $v_i < v_j$ defines constraint on the operation start times as $t_{v_i}(k) + \delta_{ij} \leq t_{v_j}(k)$ for all invocations k. The rate constraints are indicated as attributes on the operation vertices.

Operation-level delay and rate analyses are based on static path analysis of the flow graphs. We define the *length*, $\ell(G) \in Z^+$, of the longest path between the source and sink vertices assuming that each loop call is executed at least once (that is, loops are of the type 'repeat-until'). In presence of conditional paths, the length is a vector, $\underline{\ell} = (\ell[i])$ where each element $\ell[i]$ indicates the execution delay of a path in G. The elements of $\underline{\ell}$ are the lengths of the longest paths that are mutually-exclusive.

The length computation for a flow graph proceeds by a bottom-up computation of lengths from delays of individual operations. Given two operations, u and v

with delays, δ_u, δ_v, these can be composed in one of the following three ways in the flow graph:

- *Sequential composition:* The combined delay of u and v is represented by $\delta_u \odot \delta_v$ and is defined as $\delta_u \odot \delta_v \triangleq \delta_u + \delta_v$;
- *Conjoined composition:* when the operations u and v belong to two branches of a conjoined fork. A conjoined composition is denoted by $\otimes$ and the delay is defined as $\delta_u \otimes \delta_v \triangleq \max(\delta_u, \delta_v)$. A maximum of the delay is chosen to indicate that the time to completion of the concurrent operations is determined by the operation with the largest delay;
- *Disjoined composition:* when the operations u and v belong to two branches of a disjoined fork. This composition is denoted by symbol $\oplus$ and the combined delay is defined as $\delta_u \oplus \delta_v \triangleq (\delta_u, \delta_v)$.

Example 4.1. Figure 2 below shows a process graph model, G_3 and graph models on its calling hierarchy. G_3 calls G_2 that constitutes body of a loop operation, v_3. G_2 in turn calls G_1 that constitutes the body of a loop operation, v_2. Numbers in the circle indicate delay of the operations.

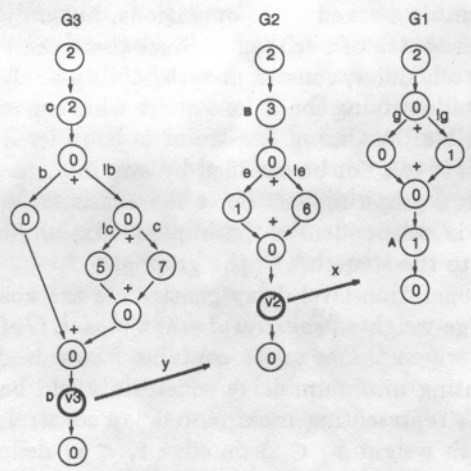

Figure 2: Example for bottom-up length computations.

For this set of graph models, the path lengths are:

$$\begin{aligned}
\underline{\ell}(G_1) &= 2 \odot (0,1) \odot (1) = (3,4) \\
\underline{\ell}(G_2) &= 2 \odot 3 \odot (1,6) \odot \underline{\ell}(G_1) = (6,11) \odot \underline{\ell}(G_1) \\
&= (9,10,14,15) \\
\underline{\ell}(G_3) &= 2 \odot 2 \odot (0,(5,7)) \odot \underline{\ell}(G_2) \\
&= (4,9,11) \odot \underline{\ell}(G_2) \\
&= (13,14,18,19,20,21,23,24,25,26)
\end{aligned}$$

□

Operation $\oplus$ and $\otimes$ are easily extended to $k-$ary operations. A disjoined composition of two delays leads to a 2-tuple delay since the two operations belong to mutually exclusive paths. This composition of delays is generalized to composition of paths as follows. In case of a sequential composition of two path lengths, $\underline{\ell}_u$ and $\underline{\ell}_v$ with cardinality n and m respectively, the resulting path length contains $n \times m$ elements, consisting of sum over all possible pairs of elements of $\underline{\ell}_u$ and $\underline{\ell}_v$. In case of a conjoined composition, the resulting path length is of cardinality $n \times m$ and consists of maximum over all possible pairs of elements. Finally, in case of a disjoined composition, the resulting path length is of cardinality $n + m$ and contains all elements of $\underline{\ell}_u$ and $\underline{\ell}_v$.

The length computations are simplified for constraint feasibility analysis since we are often interested only in the upper and lower bounds ℓ_M and ℓ_m respectively for a length vector $\underline{\ell}$ [3].

4.1 Operation-level Delay & Rate Analysis

Constraint analysis attempts to determine the existence of a schedule of operations for all possible (and conceivably infinite) values of the delay of the ND operations. If a conclusive answer is not possible, bounds on ND delays are provided to ensure constraint satisfiability. Although we do not show it here, it should be noted that not all constraints lead to a bound on ND operation delays. In particular, relative rate constraints can often give a deterministic answer to constraint satisfiability despite the presence of ND operations [13]. We state without proof the following results that form the basis of operation-level timing constraint analysis. For details and proofs the reader is referred to the references indicated.

1. Operation delay constraints are satisfiable if and only if the constraint graph is feasible and there exist no cycles with ND operations [14]. A constraint graph is considered *feasible* if it contains no positive cycle when the delay of ND operations is assigned to zero.

2. A maximum rate constraint, r_u, in G is satisfiable if $\ell_m(G) \geq \tau \cdot r_u^{-1}$ [13]. Note that minimum delay and maximum rate constraints are always satisfiable.

3. The lower bound ℓ_m used for checking the satisfaction of maximum rate constraints, also defines the fastest rate at which an operation in the graph model can be executed by a non-pipelined implementation. This points to the necessary condition for meeting a **minimum rate** constraint. A sufficient condition for meeting the minimum rate constraints is obtained by placing a bound on the time interval in invoking the concerned flow graph. In general, this bound may imply a bound on the delay due to the runtime system and/or a bound on the number of times a (calling) loop operation is invoked.

A bound on the overhead delay $\gamma_G(k)$ implies a

bound on the invocation interval of G_+, and by induction, bound on the invocation interval of all graphs in the parent hierarchy. In particular, the bound on the invocation interval of the parent process graph G_0 corresponds to a bound on the delay due to the runtime scheduler. This places restrictions on the choice of the runtime scheduler. *Note that a bound on $\gamma_G(k)$ does not imply a bound on the* latency λ *of G which may, in fact, be unbounded.* We summarize the satisfiability of a minimum rate constraint for the two cases relating to ND operations in the flow graph.

(a) A minimum rate constraint on operation, $v_i \in V(G)$, where G contains no ND operations is satisfiable if

$$\overline{\gamma}_G + \ell_M(G) \leq \frac{\tau}{r_i}$$

or equivalently, the overhead due to the runtime scheduler is bounded as follows:

$$\overline{\gamma}_o \leq \frac{\tau}{r_i} - \ell_M(G) - \sum_{G_i=G_+}^{G_o} \Delta\ell(G_i) - [\ell_m(G_o) - \ell_m(G)]$$

where $\Delta\ell(G_i) = \ell_M(G_i) - \ell_m(G_i)$. Thus a minimum execution rate constraint on a graph G that contains no ND operations is translated as an upper bound on the delay of the runtime system which is checked by comparing against $\overline{\gamma}_o$.

(b) In presence of ND operations in G, deterministic satisfiability of minimum rate constraints can only be guaranteed by transforming the the ND operations into bounded-delay operations. In practice this is accomplished by performing a context-switch in case of synchronization related operations and by bounding the number of times the body of a loop ND operation is invoked. Justification of a bound on the ND operation comes from the observation that constraint satisfiability is a property of an implementation (and not of a specification), so while it is always possible for a reactive environment to overrun any specified time bound between its actions, but that does not affect the satisfiability of an implementation. For a detailed treatment of this aspect the reader is referred to [3]. A important implication of having bounds derived from timing constraints is that it makes it possible to seek transformations to the system model which tradeoff these measures of constraint satisfiability against implementation costs.

We conclude the discussion on the tests for operation-level satisfiability analysis by an example below.

Example 4.2. Consider the hierarchy of graph models in Figure 2. We are given following constraints on operation 'A' in G_1 that constitutes loop body of operation 2 in G_2 with loop index, x, which in turn is a loop body of operation 3 in G_3:

$$r_A = 1/100 \qquad r_A^{G_1} = 1/5$$

$$r_A^{G_2} = 1/25 \qquad r_A^{G_3} = 1/50$$

Recall, that r_A^G refers to a minimum rate constraint *relative to* G. Let us first consider, $r_A^{G_1} = 1/5$ cycle^{-1}. Since this constraint is relative to G_1, therefore, there is no overhead in invocation of G_1, i.e., $\overline{\gamma}_{G_1} = 0$. Since

$$\overline{\gamma}_{G_1} + \ell_M(G_1) = 4 \leq 1/\frac{1}{5} = 5$$

Therefore, the constraint $r_A^{G_1} = 1/5$ is satisfied. Similarly, constraint $r_A^{G_2} = 1/25$ is satisfied since

$$\begin{aligned} \overline{\gamma}_{G_1} + \ell_M(G_1) &= [\ell_M(G_2) + \overline{\gamma}_{G_2} - \ell_m(G_1)] \\ &+ \ell_M(G_1) = [15 + 0 - 3] + 4 \\ &= 16 \leq 1/\frac{1}{25} = 25 \end{aligned}$$

Constraint $r_A^{G_3} = 1/50$ is satisfied since

$$\begin{aligned} \overline{\gamma}_{G_1} + \ell_M(G_1) &= [\Delta\ell(G_3) + \Delta\ell(G_2) + \overline{\gamma}_{G_3} \\ &+ \ell_m(G_3) - \ell_m(G_1)] + \ell_M(G_1) \\ &= [13 + 6 + 0 + 13 - 3] + 4 \\ &= 33 \leq 1/\frac{1}{50} = 50 \end{aligned}$$

Finally, for the minimum rate constraint $r_A = 1/100$ we should also consider the overhead $\overline{\gamma}_o$ due to the runtime scheduler which adds to the bound of 33 cycles on successive intervals of operation 'A' *relative to* G_3. Therefore, a r_A is satisfied if the delay due to the runtime scheduler is less than or equal to $100 - 33 = 67$ cycles. □

4.2 Process-level Rate Analysis

Due to the unilogic nature of the process graph, the following equation governs the time of execution of operations in the process graph:

$$t_i(k) = \max_{p_j < p_i} [t_j(k-1) + d_P(p_j, p_i)] \qquad (9)$$

Recall that $d_P(p_j, p_i)$ refers to the delay in process invocation of p_i after start of process p_j. This equation is similar to Equation 7 in case of operation-level graphs, however, it relates execution times across separate invocations of a process model.

Using the [Max, +] algebraic framework [15] the process graph can be expressed as a set of linear equations by replacing ring operations of addition and multiplication in $\Re$ by maximization and addition respectively. This algebra is useful because it allows us to reason about a system's reactive behavior using algebraic properties of its representation, in particular we use spectral properties of its matrix representation. We can rewrite

the Equation 9, using the following dot product, $\overline{\otimes}$, in [Max, +] algebra:

$$t_i(k) = [A_{i1}A_{i2}\cdots A_{in}]\overline{\otimes}\begin{bmatrix} t_1(k-1) \\ t_2(k-1) \\ \vdots \\ t_n(k-1) \end{bmatrix} \quad (10)$$

where the *process adjacency matrix*, A, of the process graph G_P, is defined as follows:

$$A_{ji} = \begin{cases} d_P(p_i, p_j) & \text{if } (p_i, p_j) \in E_P \\ \epsilon & otherwise \end{cases} \quad (11)$$

Here ϵ is the identity for max, i.e., $\epsilon = -\infty$. Note that A^l_{ij} is equal to the length of the longest path from p_j to p_i that passes through exactly $l-1$ other vertices in the process graph.

The process-level rate analysis is based on the key result that *irrespective* of the initial start times of the processes, the average rate of execution of the processes is well defined and can be efficiently computed using a graph theoretic interpretation of the eigenvalues of the process adjacency matrix [5]. The latter result is derived from a theorem due to Karp [16] that states for a matrix A with corresponding adjacency graph G_P, the eigenvalue of A is given by the maximum mean cycle weight in G_P. The mean weight of a cycle C is defined as $\frac{\sum_{e \in C} d_P(e)}{|C|}$ where $|C|$ represents the number of edges in C. A cycle is said to be *critical* if it has the maximum mean weight amongst all the cycles in the graph.

Process-level rate analysis proceeds by identification of strongly-connected components (SCCs) in the process graph. Since processes in a single SCC execute with the same average execution rate, the interesting case consists of multiple SCCs with different rates of execution. For a single SCC, the maximum mean cycle weight defines the "open-loop" average rate of execution, that is average execution ration without any synchronization relationships to other SCCs. In the context of the process-graph model where the synchronization is achieved by blocking a process execution until its predecessors have completed execution, this points to a "producer-consumer"-type synchronization relationship. The effective rate of execution for the consumer process is the lower of the "open-loop" producer and consumer rates of execution. This observation is used to determine "closed-loop" rate computation for a process graph that consists of several SCCs.

We build a *component DAG* as the graph in which there is a vertex for each SCC and an edge from u to v if and only if there is an edge from a vertex in the SCC represented by u to a vertex in the SCC represented by v in the original graph. Component DAGs are used to determine time-evolution of the the embedded system behavior in presence of inter-process synchronization relationships. We illustrate the process-level analysis by an example below.

Example 4.3.

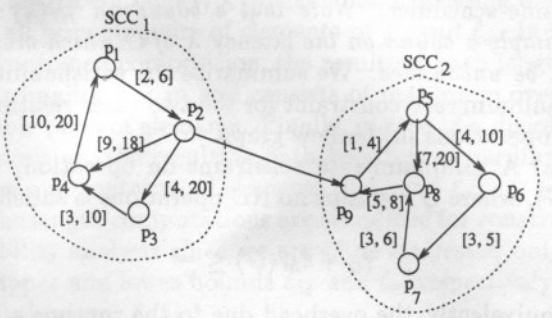

Figure 3: Process graph.

Consider the process graph shown in Figure 3 with the delay intervals as shown on the edges. For SCC_1, the lower bound on the execution rate is derived by setting edge delays at upper bound and computing the maximum mean weight cycle from the critical cycle $(p_1 \rightarrow p_2 \rightarrow p_4 \rightarrow p_1)$. Therefore, $r_l = (14.67)^{-1} = 0.068$. The upper bound on execution rate is computed by setting edge delays to the lower bound,

$$\begin{aligned} \omega_u &= \text{maximum mean weight cycle in } SCC_1 \\ &= \max\left\{\frac{10+9+2}{3}, \frac{2+4+3+10}{4}\right\} = 7 \end{aligned}$$

Thus, $r_u = (7)^{-1} = 0.142$. Hence, the rate interval for SCC_1 is $[0.068, 0.142]$.

Similarly, the "open-loop" rate interval for SCC_2 is $[0.094, 0.231]$. The execution rates in SCC_1 are not affected by the edge from SCC_1 to SCC_2, however, the rate interval for SCC_2 needs to take into consideration the rate interval of the "producer" SCC. Therefore, the actual rate interval for SCC_2 is $\min\{0.094, 0.068\}, \min\{0.231, 0.142\} = [0.068, 0.142]$.

We note that for the sake of clarity, we have computed the maximum mean cycle weight using explicit enumeration. Our implementation uses a dynamic programming algorithm to determine the critical cycles and mean weight [5]. □

5 Framework Implementation

Algorithms presented in this paper for operation-level and process-level timing constraint analysis have been implemented in the co-synthesis system, VULCAN.

The core analysis routines consist of approximately 3K lines of C-code. These routines can be used as a part of a system for constraint satisfiability analysis as well as for interactive debugging of constraint violations. The focus of operation-level analysis is to determine constraint satisfaction for a process at a time. Inter-process constraint analysis is carried out on the process graph by first computing open-loop execution rate intervals. If the computed rate intervals are contained in the intervals defined by the rate constraints, then the implementation satisfies all the rate constraints.

In case of constraint violations, debugging of the violation is done by identifying critical cycles to help the designer consider alternative implementations of the processes involved in critical cycles. If an upper bound rate constraint is violated for process p_i, then it means that p_i executes faster than required by the constraint. This situation is easier to remedy because additional delay can be introduced on some of the process graph edges to slow down the execution rate of p_i. If a lower bound rate constraint is violated, the program outputs processes on the critical cycles that lead to the constraint violation. If some of the critical cycles that cause the rate constraint violation involve self-loops, then the program also performs rate analysis after removing these self-loops. Removal of a self-loop in the process graph corresponds to pipelined implementation of the concerned process.

6 Summary and Future Work

Due to the complexity of system modeling and competing requirements of determinism in delay analysis versus non-determinism in system models from software and runtime uncertainties, an efficient decomposition of the constraint analysis problem is needed. We believe that our two-level system model provides such a decomposition where detailed delay and "open-loop" rate analysis can be carried out at the individual process-level. However, the effect of inter-process synchronizations is best handled at the process level which is amenable to algebraic analysis because of the (unilogic) modeling restrictions. Corresponding to our model of the timing constraints, the constraint analysis is divided into operation and process levels. The operation-level analysis focus is on individual processes, whereas process-level analysis evaluates constraint satisfiability in view of inter-process synchronizations. Both analyses begin by identifying cases where the imposed constraints are inconsistent, that is, these can not be satisfied by any implementation of the embedded system, and proceed to identify operation and process bottlenecks in the form of ND operations and critical loops that cause a particular constraint violation. Extensive designer input is needed during this process to explore alternative process implementations such as the eventual implementation satisfies the constraints.

To complement this analysis, we plan to develop algorithms for runtime evaluation for embedded systems and its integration with detailed timing analysis.

7 Acknowledgments

The author would like to thank Anmol Mathur for discussions and contributions. This research was supported by a grant from the AT&T Foundation and a grant from NSF No. MIP 95-01615.

References

[1] W. Wolf, "Hardware-Software Co-design of Embedded Systems," *IEEE Proceedings*, vol. 82, no. 7, pp. 965–989, July 1994.

[2] D. Ku and G. D. Micheli, *High-level Synthesis of ASICs under Timing and and Synchronization Constraints.* Kluwer Academic Publishers, 1992.

[3] R. K. Gupta, *Co-Synthesis of Hardware and Software for Digital Embedded Systems.* Kluwer Academic Publishers, Boston, 1995.

[4] V. Cerf, *Multiprocessors, Semaphores and a Graph Model of Computation.* PhD thesis, UCLA, Apr. 1972.

[5] A. Mathur and R. K. Gupta, "Rate analysis for embedded systems," (submitted. available as technical report), Computer Science, University of Illinois, 1995.

[6] H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Reading, MA: Addison-Wesley, 1978.

[7] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Method of Validating Them," *IEEE Trans. Software Engineering*, vol. SE-11, no. 1, pp. 80–86, Jan. 1985.

[8] R. Camposano and A. Kunzmann, "Considering Timing Constraints in Synthesis from a Behavioral Description," in *Proc. ICCCD*, pp. 6–9, 1986.

[9] R. Lauber, "Forecasting Real-Time Behavior During Software Design Using a CASE Environment," *Journal of Real-Time Systems*, vol. 1, no. 1, pp. 61–76, June 1989.

[10] D. W. Leinbaugh, "Guaranteed Response Times in a Hard Real-Time Environment," *IEEE Trans. Software Engg.*, vol. SE-6, no. 1, pp. 85–91, 1980.

[11] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, 1982.

[12] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[13] R. K. Gupta and G. D. Micheli, "Specification and Analysis of Timing Constraints for Embedded Systems," (submitted. available as tech. report), University of Illinois, 1995.

[14] D. Ku and G. D. Micheli, "Relative Scheduling Under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits," *IEEE Transactions on CAD/ICAS*, vol. 11, no. 6, pp. 696–718, June 1992.

[15] F. Bacelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity.* John Wiley and Sons, New York, 1992.

[16] R. M. Karp, "A characterization of the minimium cycle mean in a digraph," *Discrete Math.*, vol. 23, pp. 309–311, 1978.