Synthesis of Interface Controllers from Timing Diagram Specifications

Abdelhalim El-Aboudi, El-Mostapha Aboulhamid, Eduard Cerny
Labratoire LASSO, Département IRO, Université de Montréal
Montréal (Québec) CANADA

Abstract
We present a method for verifying the realizability of a timing diagram, ensuring the synthesis of the underlying interface is possible. If necessary, a heuristic is introduced to render explicit hidden timing constraints implied by the specification. A relative schedule of output events is computed, accepting input events from the complete timing space defined by the assumed constraints on the environment.

Keywords
Timing diagrams, timing constraints, realizability, relative scheduling, synthesis.

1 Introduction

Interface design is an important aspect of the design of digital microelectronic systems. This importance is growing with the complexity of digital circuits. In order to manage this complexity, hardware systems are designed as distributed systems which require well defined interactions between the different components. The communication protocol between components is characterized by temporal constraints, hence the proper timing of the interface controllers is crucial. Because of their simplicity, and expressiveness, timing diagrams (TDs) are used to specify the behavior of interfaces. Our work deals with the realizability of TD specifications, and the subsequent synthesis of interface controllers. Maximum separation[2,3,4,5,6,14], consistency[6], satisfiability[6], realizability and causality[1] are among the most important concepts developed in the literature on TDs with quantitative timing constraints. Maximum separation determines temporal distances between events in the TD, and it constitutes the basic computation on TDs that is used in other aspects of analysis. Consistency ensures that the given system of constraints has at least one solution. Compatibility verifies whether devices built according to their TD specifications can correctly interact when connected together. Authors of [1] define causality property as a sufficient condition of realizability and give a more elaborate solution to compatibility based on causality. In this work we present a new method for determining whether a TD is realizable. Our method can deal with concurrent constraints between input and output events not allowed in [1], and no causal partition over events is necessary, thus avoiding the complexity of determining causal partitions as in [16]. The relative scheduling used in our method constitutes a generalization of relative scheduling, since the assume constraints can be bounded compared to unbounded only in [10]; we Also allow intermixing of input and output events which is the case in TD specifications.

Many previous approaches dealt with interface synthesis. The method of Nestor and Thomas [8] is based on behavioral synthesis, is limited to synchronous interfaces with linear constraints and might lead, for complex interfaces, to tedious specifications. Furthermore, the method does not determine the adequate clock rate automatically. The well known work by Borriello [7], based on templates, has its own limitations. It deals only with linear constraints, and may gives rise to race conditions. Another approach is described in [9,11] using process calculus. This method considers precedences without quantitative temporal constraints. The recent method reported in [12], developed for embedded systems, is limited to the derivation of a combinatorial interface transducer which ensures the connection, either by direct wires or via a combinatorial circuit, between the ports of communicating circuits. In this work, we present an approach to the synthesis of general interface controllers from TD specifications with linear temporal constraints. It consists of deriving a new timing diagram from the original one, containing all the given constraints on input events, and all the initial constraints and some new constraints on the output events such that the schedule of each output event depends only on its immediate parents.

2 Background

2.1 Interface specifications

An *interface* consists of a set of channels called *ports* serving to exchange information between a system and its environment, a set of rules defining a *protocol of communication*, and timing relationships between *events* occurring on the ports. The interface behavior can be specified using timing diagrams (TDs). An *event graph EG* can be associated with each TD: $EG = (E,C)$ where the set of vertices E corresponds to the set of events and the set of directed edges C corresponds to a set of constraints $C = \{c_{ij} = (e_i, e_j, [l_{ij}, u_{ij}]) \mid e_i, e_j \in E\}$. To each event e_i, we assign an *occurrence time* of e_i denoted by $t(e_i)$ such that $l_{ij} \le t(e_j) - t(e_i) \le u_{ij}$ *for all* $c_{ij} \in C$. Occurrence time as well as upper-bounds are supposed to be non-negative reals. c_{ij} is a *precedence constraint* if $l_{ij}.u_{ij} > 0$. Otherwise, it is a *concurrent constraint*. A constraint is *explicit* if it figures in C; it is *implicit* if it can be deduced by some computation (e.g. maximum separation). For a constraint $c_{ij} = (e_i, e_j, [l_{ij}, u_{ij}])$, e_i is called *parent* of e_j. An event (node) is said to be a *convergence event* (*node*) if it has more than one parent.

A direction is associated with each event: input or output.

Denote by I the set of all input events, and O the set of all output events. $E = I \cup O$ and $I \cap O = \varnothing$. A timing constraint $c_{ij} = (e_i, e_j, [l_{ij}, u_{ij}])$ is a *commit constraint* if $e_j \in O$, otherwise it is an *assume constraint*. We denote by A (respectively K) the set of assume (commit) constraints over E, $A \cup K = C$. A commit constraint is under the control of the designer, since it concerns an output event to be produced by the system under construction. An assume constraint is guaranteed by the environment, and we cannot force any specific separation time between events e_i and e_j. We denote by *CS-G*, *CS-A* and *CS-K* the system formed by all timing constraints in C, the assume constraints in A and the commit constraints in K, respectively.

3 Timing analysis

Analysis of the timing behavior of the interface is crucial for two purposes. First, for interface verification, to check if the implemented circuit satisfies all timing requirements, so that all output events will be produced within the time interval required and expected by the circuit's environment. Second, for synthesizing digital circuits, to determine delays within which output events must be produced.

3.1 Maximum separation

A *separation time* is the difference between the occurrence times of a pair of events: $(s_{ij} = t(e_j) - t(e_i))$.The computation of the maximum separation times between events in a timing diagram does not take into account the nature of events (input or output) but only the system of constraints *CS-G*. Several algorithms have been developed for computing the maximum separation. The complexity of these algorithms depends on the type of the timing constraints allowed[2][3][6,7][13][14].

An event graph is said to be *tight* if the bounds of each constraint correspond exactly to the maximum separation on the whole event graph.

3.2 Consistency

Definition 1 [6] An event graph *(E, C)* is consistent if and only if the set of n-tuples $(t(e_1), \ldots, t(e_n))$ satisfying C is not empty.

Note that consistency does not take into account the nature of the events. This does not guarantee that a given specification is implementable. In the example of Figure 1, we can assign an occurrence time to each event such that all given constraints are satisfied. For example, $(t(i_1), t(i_2), t(o_1), t(o_2)) = (0, 10, 10, 30)$ is a solution. But we cannot find any possible assignment when the environment produces the event i_2 at time 100 after i_1 which it is free to do.

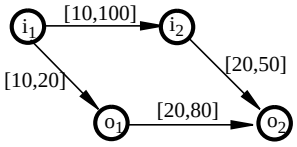

Figure 1 Example of consistent but not realizable event graph

4 Realizability

Let $EG = (E, C)$ be an event graph, $C = A \cup K$. $|A| = m$. Let $\boldsymbol{e} = (e_1, \ldots, e_k)$ be a tuple of events in E, and denote by $\boldsymbol{t(e)}$ the vector of occurrence times $(t(e_1), \ldots, t(e_k))$. Let O_s be a set of all output events which constitute the source events for constraints in A, $O_s = \{e_i \in O \mid c_{ij} \in A\}$, $|Os| = q$. For each constraint $c_{ij} \in A$ (respectively K), we write $\delta_{ij} = t(e_j) - t(e_i)$ $(\gamma_{ij} = t(e_j) - t(e_i))$. The interval $[l_{ij}, u_{ij}]$ is denoted by I_{ij}, we have $\delta_{ij} \in I_{ij}$ for linear constraints. We denote by δ the vector of δ_{ij} corresponding to all c_{ij} in A.

Definition 2 A function f from $(R^+)^n$ to R^+ is a causal function if and only if it is a constant function or for each vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in (R^+)^n$ there exist a variable x_i in $\boldsymbol{x}$ such that $f(\boldsymbol{x}) \geq x_i$.

Examples of causal functions are the functions *min* and *max*.

Definition 3 A function h from O_s to R^+ is causal if there exists q causal functions f_k from $(R^+)^m$ to R^+such that for each event $o_k \in O_s$, (k = 1, …, q), we have $h(o_k) = f_k(\boldsymbol{t(i)})$ where $\boldsymbol{i} = (i_1, \ldots, i_m)$.

The space of the occurrence times of the input events which respect the assume constraints may depend on the occurrence times chosen for the output events in O_s. Hence the possible values of δ_{ij} depend on these choices. Given that $\boldsymbol{t(o)} = \boldsymbol{h(o)}$, where $\boldsymbol{h(o)} = (h(o_1), \ldots, h(o_q))$, we denote by S_h the space of all possible values of the vector δ. $S_h = \{\delta \in \prod_{ij} I_{ij}$ for $c_{ij} \in A$ *| CS-A is consistent}.*

Definition 4 An event graph $EG = (E, C)$ is said to be *realizable* if and only if: There exists a causal function h from O_s to R^+ such that $\boldsymbol{t(o)} = \boldsymbol{h(o)}$, $\boldsymbol{o} = (o_1, \ldots, o_q)$ the vector of events in O_s with $S_h \neq \varnothing$, and $\forall\ \delta \in S_h$, the system (*CS-K*) is consistent.

Example of realizable TD: Consider the event graph of Figure 2 (without dashed edges). We have:

CS-A = $(10 \leq \delta_1 \leq 20) \wedge (10 \leq \delta_2 \leq 30) \wedge (t(i_2) = t(o_3) + \delta_1) \wedge (t(i_3) = t(o_4) + \delta_2)$.

CS-K = $(10 \leq \gamma_1 \leq 60) \wedge (20 \leq \gamma_2 \leq 50) \wedge (10 \leq \gamma_3 \leq 30) \wedge (10 \leq \gamma_4 \leq 30) \wedge (10 \leq \gamma_5 \leq 20) \wedge (40 \leq \gamma_6 \leq 60) \wedge (20 \leq \gamma_7 \leq 70) \wedge (t(o_1) = t(i_1) + \gamma_1) \wedge (t(o_2) = t(i_1) + \gamma_2) \wedge (t(o_3) = t(o_1) + \gamma_3) \wedge (t(o_4) = t(o_1) + \gamma_4) \wedge (t(o_4) = t(o_2) + \gamma_5) \wedge (t(o_5) = t(i_2) + \gamma_6) \wedge (t(o_5) = t(i_3) + \gamma_7)$. $O_s = (o_3, o_4)$.

Let us choose the function h such that: $t(o_3) = t(i_1) + 90$ and $t(o_4) = t(i_1) + 70$. $S_h = \{\delta \in \prod_{ij} I_{ij}$ for $c_{ij} \in A$ *| CS-A is con-*

sistent} = [10, 20] × [10, 30].
We must now verify if for all $\delta \in S_h$ we can find γ_{ij} such that *CS-K* is satisfied. If we take $\gamma_1 = 60$, $\gamma_2 = 50$, $\gamma_3 = 40$, $\gamma_4 = 10$, $\gamma_5 = 20$, we get $t(o_5) = t(i_1) + \delta_1 + 100 + \gamma_6 = t(i_1) + \delta_2 + 70 + \gamma_7$. We must choose γ_6 and γ_7 such that $\gamma_7 - \gamma_6 = \delta_1 - \delta_2 + 30 \in [10, 40]$. For all $\delta \in S_h$, we can always find $\gamma_7 \in [30, 70]$ and $\gamma_6 \in [40, 60]$ such that *CS-K* is consistent. So the event graph *EG* is realizable.

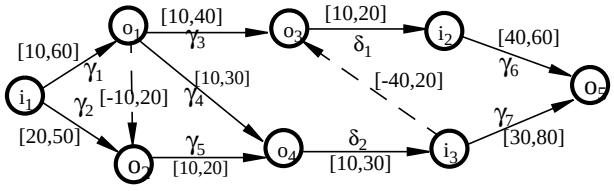

Figure 2 Example1

4.1 Verification of realizability

Firing of an event *z*, by either the environment or the system, is not possible unless all constraints affecting *z* are satisfied and this is for all the possible occurrence times of past events. *Local consistency* (defined in the following subsection) is verified for all convergence nodes. If it does not hold, we either suggest corrections to the system or tighten commit constraints to obtain the local consistency property.

A. Local consistency

We suppose that all constraints are tight. The set of constraints induces a partial order on the events, from which we can obtain a total order. Let us consider the last convergence node *z* according to this total order with two parents e_1 and e_2 and constraints $(e_1, z, [m_1, M_1])$, $(e_2, z, [m_2, M_2])$**.** The firing time of *z* must satisfy: $m_1 \le t(z) - t(e_1) \le M_1$ and $m_2 \le t(z) - t(e_2) \le M_2$. This implies the following condition on the separation time s_{12} between e_1 and e_2:

$$\boldsymbol{m_1 - M_2 \le s_{12} = t(e_2) - t(e_1) \le M_1 - m_2} \qquad (1)$$

Definition 5 (Local consistency): Let $EG = (E, C)$ be an event graph. Let *z* be the last node of *EG*, and $P(z)$ a set of its parents in *EG*. Let $EG' = (E', C')$, where $E' = E \backslash \{z\}$, and $C' = C \backslash \{(e_i, z, [l, u]) \in C\}$. *z* is *locally consistent* if and only if $P(z)$ is a singleton or $\forall\, e_1, e_2 \in P(z)$ the maximum separation time s_{12} of e_1, e_2 (respectively s_{21} of e_2, e_1) computed over *EG'* is less than or equal to the maximum separation time s_{12} (s_{21}) computed over $\{e_1, e_2, z\}$. *EG* is said to be locally consistent if the last event *z* and the sub-graph *EG'* are locally consistent.
Example: Consider the event graph *EG* of Figure 2, o_5 is the last event and is not locally consistent. The parents of o_5 are (i_3, i_2), we get $s_{32} = 40$ and $s_{23} = 40$ computed over *EG'*, whereas from the equation (1) we should have $s_{32} = 40$ and $s_{23} = 30$.

B. Realizability and local consistency relationship

In this section we study the relationship between the local consistency of event graphs and the realizability of timing diagrams.
Definition 6 (Constrained Event Graph): An event graph $EG_1 = (E_1, A_1 \cup K_1)$ is a constrained event graph (*CEG*) of an event graph $EG_2 = (E_2, A_2 \cup K_2)$ if $E_1 = E_2$, $A_1 = A_2$, $K_1 = \{K_2$ *with restricted intervals*$\} \cup \{$*set of additional commit constraints*$\}$.
Theorem: (Realizability of EGs): An event graph *EG = (E, C)* is realizable if and only if there exists at least one locally consistent constrained event graph *CEG associated with* it.

C. Finding a locally consistent constrained event graph

In case when an *EG* does not verify local consistency, some judicious modifications can be done to the commit constraints to make it locally consistent. The problem is thus reduced to determining which commit constraint to modify and/or to add, without altering the given assume constraints nor causing any timing inconsistency. Figure 2 illustrates the different possibilities. o_5 does not verify the local consistency property, we should enforce the implicit assume constraint between i_3, i_2 to be in [-30, 40]. To obtain this condition we look among the events in the graph for a candidate new commit constraint which could be in this case (i_3, o_3, [-40, 20]). Suppose without loss of generality, that the implicit assume constraint to be enforced is between e_1 and e_2 with interval [m, M], and an output event o_k is a parent of e_2 with a constraint $(o_k, e_2, [m_2, M_2])$ such that the commit $(e_1, o_k, [l, u])$ is to be added, then such commit should verify the following condition:

$$\boldsymbol{u + M_2 \le M \text{ and } m \le l + m_1} \qquad (2)$$

By examining recursively the different sub-graphs, we obtain the sub-graph containing $\{i_1, o_1, o_2, o_4\}$. o_4 is not locally consistent, but we can add a commit constraint $(o_1, o_2, [-10, 20])$ to ensure it.

Algorithm for finding a locally consistent event graph:

Step1: tighten the event graph
/* warnings are generated on each eventual modification of assume constraints*/
Step2: sort the list of convergence nodes in a reverse topological order
Step3:
for each convergence node
{determine its parents
for each pair of its parents ***do recursively*** /* by a depth-first search */
{verify the local consistency condition
if the condition holds ***then*** continue

else if a commit constraint can be added ***then*** {add the commit constraint and update the list of convergence nodes}
else {find a commit constraint satisfying equation (2)
update the list of convergence nodes}
}
}

5 Synthesis and Experiments

When the local consistency property holds for all convergence nodes, the next step is to compute a relative schedule for all output events. It can be shown that ALAP relative schedule [10] with respect to the parents is a realizable schedule. For the example of Figure 2 (with the dashed edges included). A realizable schedule is derived as follows:

$t(o_1) \in [10, 60]$,
$t(o_2) = \min(t(i_1) + 50, t(o_1) + 20)$,
$t(o_3) = \min(t(o_1) + 40, t(i_3) + 10)$,
$t(o_4) = \min(t(o_2) + 20, t(o_1) + 30)$,
$t(o_5) = \min(t(i_2) + 60, t(i_3) + 80)$.

The algorithm for local consistency was implemented using CLP(BNR) Prolog [15] which is a constraint logic programming system. We have done experiments with a number of interface specifications such as the interface of the *Z84C0008 CPU* memory write cycle (Figure 3). The clock cycle used is of 125ns. The event graph established for the write operation, without wait signal, have 13 vertices and 21 edges. We get a constraints system with 5 assume and 16 commit constraints. A commit constraint must be added between i_3 and o_7 with an interval of [- ∞, 55].

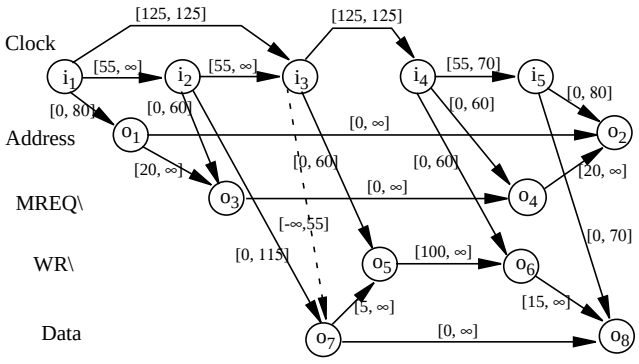

Figure 3 Memory write TD for Z84C0008 CPU

The following ALAP relative schedule for the output events can be used to implement the controller:
$t(o_1) = t(i_1) + 80$, $t(o_2) = t(i_5) + 80$, $t(o_3) = t(i_2) + 60$, $t(o_4) = t(i_4) + 60$, $t(o_5) = t(i_3) + 60$, $t(o_6) = t(i_4) + 60$, $t(o_7) = \min(t(i_2) + 115, t(i_3) + 55)$, and $t(o_8) = t(i_5) + 70$.
The relative scheduling is given in time units. A method similar to [16] can be used to produce schedules in terms of clock cycles.

6 Conclusion

We presented a heuristic method to determine if a timing diagram is realizable. It is based on checking the satisfaction of local consistency property for all events in the timing diagram. This technique can be used to generate a proposition of modification for a non realizable timing diagrams. We derived a synthesis method for interface controllers starting from timing diagram specifications with linear constraints. This method is based on a relative scheduling of output events from its immediate parents. Such a method of scheduling has the advantage of generating minimum offset delays for ALAP scheduling, and it constitutes a generalization of relative scheduling.

References

[1] E. Cerny, K. Khordoc, "*Semantics and Verification of Action Diagrams with Linear Timing Constraints*", Transactions on Design Automation of Electronic Systems, Vol.3, No.1, Jan'98.

[2] K.McMillan, D.L.Dill, "*Algorithms for Interface Timing Verification*", in Proceedings, IEEE ICCD'92, pp.48-51.

[3] T.-Y.Yen, A.Ishii, A.Casavant, W.Wolf, "*Efficient Algorithms for Interface Timing Verification*", in Proceedings, the Euro-DAC'94.

[4] H.Hulgaard, S.M.Burns, T.Amon, G.Borriello, *"An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems*", IEEE Transactions on Computers. Vol.44, No.11, Nov'95, pp.1306-1317.

[5] T.Amon, H.Hulgaard, G.Borriello, S.Burns, "*Timing Analysis of Concurrent Systems: An Algorithm for Determining Time Separation of Events*", in Proceedings, IEEE ICCD'93.

[6] J.A.Brzozowski, T.Gahlinger, F.Mavaddat, "*Consistency and Satisfiability of Waveform Timing Specifications*", Networks, Vol.21, 1991, pp.91-107.

[7] G. Borriello, "*A New Interface Specification Methodology and its Application to Transducer Synthesis*", Ph.D. Thesis, EECS, University of California,Berkeley, 1988.

[8] J.A. Nestor, D.E. Thomas, "*Behavioral Synthesis with Interfaces*", IEEE ICCAD'86, pp.112-115.

[9] W.-D. Tiedemann, *"An Approach to Multi-paradigm Controller Synthesis from Timing Diagram Specifications"*, in Proceedings, the Euro-DAC'92, pp.522-527.

[10] D. Ku, G. De Micheli, "*Relative Scheduling under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits*", IEEE Transactions on Computer-Aided Design, Vol.11, No.6, June 1993, pp.696-718.

[11] W.-D.Tiedemann, "*Bus Protocol Conversion: from Timing Diagrams to State Machines"*, EuroCAST'91, pp.365-377.

[12] K.-S.Chung, R.K.Gupta, C.L.Liu, "*An Algorithm for Synthesis of System-Level Interface Circuits*", in Proceedings, IEEE ICCAD'96, pp.442-447.

[13] P.Vanbekbergen, G.Goossens, H. De Man, *"Specification and Analysis of Timing Constraints in Signal Transition Graphs"*, in Proceedings, the European Conference on Design Automation, 1992, pp.302-306.

[14] P. Girodias, E. Cerny and W.J. Older, "*Solving Linear, Min and Max Constraint Systems Using CLP based on Relational Interval Arithmetic*". In Theoretical Computer Science, CP'95 Special Issue, Volume 173. Forthcoming February 1997.

[15] W.J. Older, F. Benhamou, "*Programming in CLP (BNR)*". In PPCP'94, Newport, RI(USA), 1993.

[16] E.Cerny, Y. Wang, M.Aboulhamid, "*Discrete-Time Scheduling under Real-Time Constraints*", IWLAS'97.

Acknowledgments: The work was partially supported by a Micronet Grant No. S.4.MCI.