Hardware Timing Verification using KRONOS *

Oded Maler Sergio Yovine

SPECTRE – VERIMAG
Miniparc-ZIRST, 38330 Montbonnot, France
E-mail: {Oded.Maler,Sergio.Yovine}@imag.fr

Abstract

In this paper we describe the KRONOS *system, a tool for verifying real-time properties based on the model of timed-automata. As an example, we show how* KRONOS *is applied to the verification of a MOS circuit under various delay assumptions.*

1 Introduction

The use of finite-state automata as models of synchronous circuits is as old as computer science. Recently, verification methods for finite-state systems have proliferated from academia to industry with the progress in efficient techniques for symbolic model-checking [10], [3]. Timed automata, that is, automata augmented with fictitious clocks that can measure the time between various events, were introduced by Dill and Alur [7], [2] as models for real-time systems and asynchronous circuits.

At this real-time level of modeling, systems are not viewed anymore as generators of sequences over an abstract time domain, but rather as generators of signals over the real time domain. This allows us to speak *quantitatively* about the implementation (the time it takes for an instruction to be performed or for a gate to switch), the environment (the minimal time between two requests) and the requirements (the response time of the system).

The KRONOS system [4], [5], developed at VERIMAG during the last five years, represents the state-of-the-art in verification tools for real-time. It has been applied in the past to the verification of a variety of time-dependent protocols.

In this paper we illustrate the application of KRONOS in another domain, namely timing analysis of hardware circuits. The theoretical basis of the translation of Boolean asynchronous circuits with delay uncertainties into timed automata has been presented in [11]. Here we apply this methodology to the verification of a MOS circuit, a problem posed to us by members of IBM Haifa research group.

The paper is organized as follows. In section 2 we give a short introduction to timed automata and real-time logics which constitute the theoretical infrastructure for KRONOS. In section 3 we describe the KRONOS system and in section 4 we demonstrate how MOS circuits are modeled as timed automata and verified using KRONOS.

2 Theoretical Background

2.1 Timed Automata

A timed automaton $\mathcal{A}$ is a tuple $\langle \mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mu \rangle$. $\mathcal{S}$ is a finite set of *locations*. $\mathcal{X}$ is a finite set of real-valued variables called *clocks* whose values increase uniformly with time. $\mathcal{L}$ is a finite set of *labels*. $\mathcal{E}$ is a finite set of *edges*. Each edge e is a tuple (s, L, ψ, ρ, s') where $s, s' \in \mathcal{S}$, $L \subseteq \mathcal{L}$, ψ is a linear constraint on the values of the clocks expressing the timing condition requiered for the transition to be taken, and $\rho \subseteq \mathcal{X}$ is the set of clocks to be reset to 0 by the transition. $\mathcal{I}$ is a function that associates a constraint on clocks to each location. μ is a labeling of the locations with a set of atomic propositions over $\mathcal{P}$. Readers who wish at this point to see a concrete example are invited to jump to the beginning of section 4.

*In Proc. 7th Israeli Conference on Computer Systems and Software Engineering, June 12-13, 1996, Herzliya, Israel.

A *state* of $\mathcal{A}$ is a pair (s, v) where v is a clock valuation. The state (s, v) has a *discrete transition* to (s', v'), denoted $(s, v) \rightarrow^{e} (s', v')$, if v satisfies the constraint ψ labeling the edge e and v' is such that $v'(x) = 0$ if $x \in \rho$, otherwise $v'(x) = v(x)$. That is, discrete transitions consist in moving to another state by taking an edge whose condition is satisfied by the values of the clocks. The state (s, v) has a *time transition* of duration t to (s, v'), denoted $(s, v) \rightarrow^{t} (s, v')$, if $v'(x) = v(x) + t$ for all $x \in \mathcal{X}$, and for all t', $0 \leq t' \leq t$, $v + t'$ satisfies the invariant $\mathcal{I}(s)$. That is, time transitions consist in letting time progress without changing the control location. A *run* $r \in \mathcal{R}$ of $\mathcal{A}$ is a sequence $q_0 \rightarrow^{t_0} q'_0 \rightarrow^{e_0} q_1 \ldots$ such that $\sum_{i \geq 0} t_i$ diverges.

2.2 Symbolic Verification

KRONOS implements a symbolic verification method based on predicate transformers for computing the set of predecessors (backward analysis) and successors (forward analysis) of sets of states symbolically represented as disjunctions of convex polyhedra. A more detailed description of the symbolic verification method can be found in [9].

Backward analysis. Given a set of states $Q \subseteq \mathcal{Q}$ and an edge $e \in \mathcal{E}$, the predecessors of Q by e are those states that can reach a state in Q by letting time elapse for some amount and then moving through e. We define:

$$pred_e(Q) \quad = \quad \{q' \mid \exists q \in Q, t \in \mathsf{R}^+ . \, q' \rightarrow^{t} \rightarrow^{e} q\}.$$

The set of all predecessors of Q is $pred(Q) = \bigcup_{e \in \mathcal{E}} pred_e(Q)$.

Forward analysis. Given a set of states $Q \subseteq \mathcal{Q}$ and an edge $e \in \mathcal{E}$, the successors of Q by e are those states that can be reached from a state in Q by a time transition followed by a discrete transition through e. We define:

$$post_e(Q) \quad = \quad \{q' \mid \exists q \in Q, t \in \mathsf{R}^+ . \, q \rightarrow^{t} \rightarrow^{e} q'\}.$$

The set of all successors of Q is $post(Q) = \bigcup_{e \in \mathcal{E}} post_e(Q)$.

2.3 The logic TCTL

TCTL [1, 9] is an extension of the temporal logic CTL [8] that allows reasoning about the quantitative real-time elapsed between events along the runs of a timed automaton. The formulas of TCTL are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists\Diamond_I \varphi \mid \forall\Diamond_I \varphi$$

where $p \in \mathcal{P}$ is a basic predicate and I is an interval. I may be open or closed, bounded or unbounded. Intuitively, $\exists\Diamond_I \varphi$ means that there exists a run and a state where φ holds and the timed elapsed until this state since the initial state of the run belongs to the interval I. The formula $\forall\Diamond_I \varphi$ means that for all runs the above property holds. We write $\forall\Box\varphi$ for $\neg\exists\Diamond\neg\varphi$ meaning that φ holds in all states of all runs. The formal semantics of TCTL is given in [1, 9].

Model-checking consists in verifying whether a timed automaton $\mathcal{A}$ satisfies a TCTL-formula φ. Model-checking can be done symbolically using the backward and forward verification methods. We illustrate here how the methods work for only two of the temporal operators. For more details see [9].

Suppose we want to verify $q_{init} \Rightarrow \exists\Diamond Q$, that is, if a state in Q is reachable from the initial state q_{init}. The backward-computation algorithm is as follows: start with $Q_0 = Q$, and for $i \geq 0$ compute $Q_{i+1} = Q_i \cup pred(Q_i)$. If $q_{init} \in Q_i$ for some i, the answer is "YES", otherwise, the answer is "NO". This procedure always terminates. A similar algorithm is used to verify $q_{init} \Rightarrow \exists\Diamond_I Q$.

To check that the system, always remains at a set of states Q we verify the formula $q_{init} \Rightarrow \forall\Box Q$, or equivalently, $\neg(q_{init} \wedge \exists\Diamond\neg Q)$, that is, there is no run starting at q_{init} that reaches a state outside Q. On this formula, the forward-computation algorithm works as follows: start with $Q_0 = \{q_{init}\}$, and for $i \geq 0$ compute $Q_{i+1} = Q_i \cup post(Q_i)$. If $Q_i \cap \neg Q \neq \emptyset$ for some i, the answer is "NO", otherwise, the answer is "YES". Notice that during the computation we can keep track of the transitions taken, so as when the answer is "NO" we can exhibit a counter-example, that is, a sequence of transitions that leads from q_{init} to a state in $\neg Q$.

3 The Tool KRONOS

KRONOS [4, 5] is a tool developed with the aim to assist the user to validate complex real-time systems. The tool checks whether a real-time system modeled by a timed automaton $\mathcal{A}$ satisfies a timing property specified by a TCTL-formula φ. Figure 1 illustrates the structure of the tool.

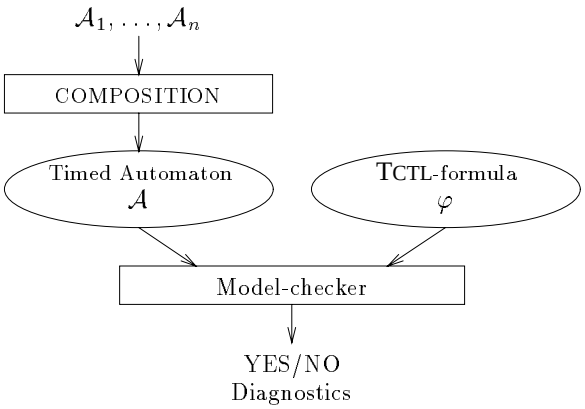

Figure 1. The KRONOS system.

A system is usually described as a set of timed automata that run in parallel and synchronise via the names labeling the edges. The parallel composition of timed automata is defined in [5]. KRONOS takes as input a set of files each one describing a single component and computes their product which is itself a timed automaton. The latter as well as the TCTL-formula to be checked are fed into the model-checker that answers "YES" whenever the formula is verified, otherwise it answers "NO" exhibiting a counter-example. The syntax of the input (automata, formulae) and output (counter-example) files, is demonstrated via examples in the appendices.

KRONOS has been used to verify a variety of protocols whose correctness strongly depends on timing parameters. Some examples of benchmarks verified with KRONOS are the Fischer's mutual exclusion protocol [6], the Tick-Tock [4] and the FDDI [6] communication protocols, and the Philips audio control protocol [5].

4 Circuit Timing Analysis

In this section we show how MOS circuits are modeled by a timed automata and how they are verified using KRONOS. This is *not* intended to be a complete nor accurate introduction to VLSI, but rather an illustration of the modeling principles.

A MOS transistor (figure 2) is a three-terminal device. It can be in two basic states *On* and *Off*. When it is *On*, the current flows between terminals X_1 and X_2. The transition between the two states is controlled by the switching terminal S. When the transistor is *Off* and S goes up (we denote this event by $S\uparrow$), after a delay of 1 time unit the state of the transistor becomes *On*. When S switches back, the gate goes to *Off* immediately. The modeling of such a transistor as a timed automaton is rather straightforward: The transition from an *Off* to *Rising* state is triggered by the event $S\uparrow$. This transition resets the clock C to zero. If an event $S\downarrow$ arrives during the interval $[0,1]$ the automaton goes back to *Off*. Otherwise, when $C = 1$ is goes to *On* (emitting the signal $T\uparrow$) and stays there until the next $S\downarrow$ event.

Remark: We can employ other transistor models as well. For example, if a non-zero delay is associated with falling of the transitor, we would need a 4-state automaton with the additional state *Falling*. We can also model uncertainties in delays. The formal translation of general Boolean asynchronous circuits into timed automata is described in [11].

The circuit we consider appears in figure 3. It consists of 8 transistors and 4 input wires P, A, B, and C which behave as follows:

- P goes high immediately at start, then goes down after 20 time units, goes up again after 8 time and restarts again.
- A can rise between 0 and 6 time units after start (i.e. after P goes up) and falls down 16 units later.
- B rises 3.3 units after A and falls 10 units later
- C rises between 1 and 3 units after start and falls between 23.5 and 24.9 units after start.

The transistors T_1, T_5 and T_6 switch to *On* by the *falling* of their input signals. The value on the X wire is *On* if current flows thru T_1 but not leaking via T_2 and T_3 or T_4. The input signals are modeled by the four timed automata in figures 4–7. State staying conditions (invariants) are written inside the states. Transitions are labeled by conditions (if not trivial), by event labels and by sets of clocks tobe reset. Note that we allow non-determinism in the behavior of the inputs. For example, when C_A is in the interval $[0,6]$ A can either stay at A_0 or move to A_1 and generate an $A\uparrow$ transition. An example of the KRONOS source files for the input A and its transistor T_2 appears in appendix 1.

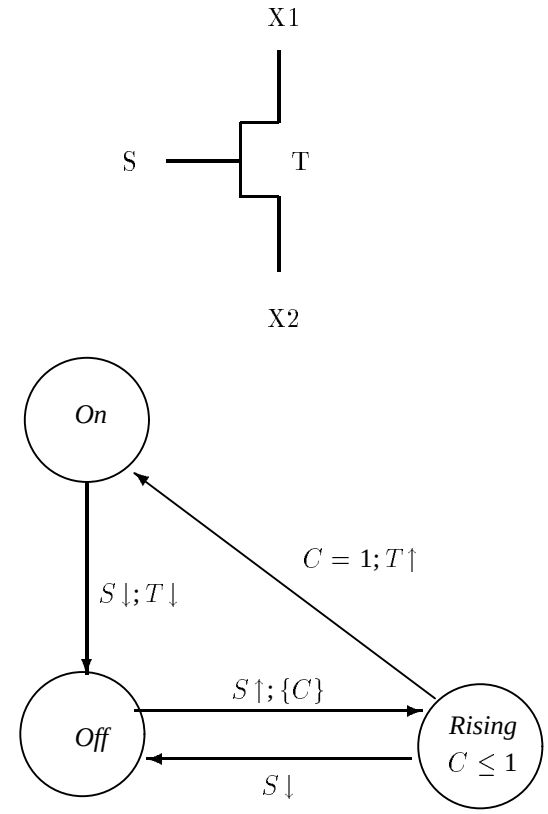

Figure 2. A transistor and its associated timed automaton.

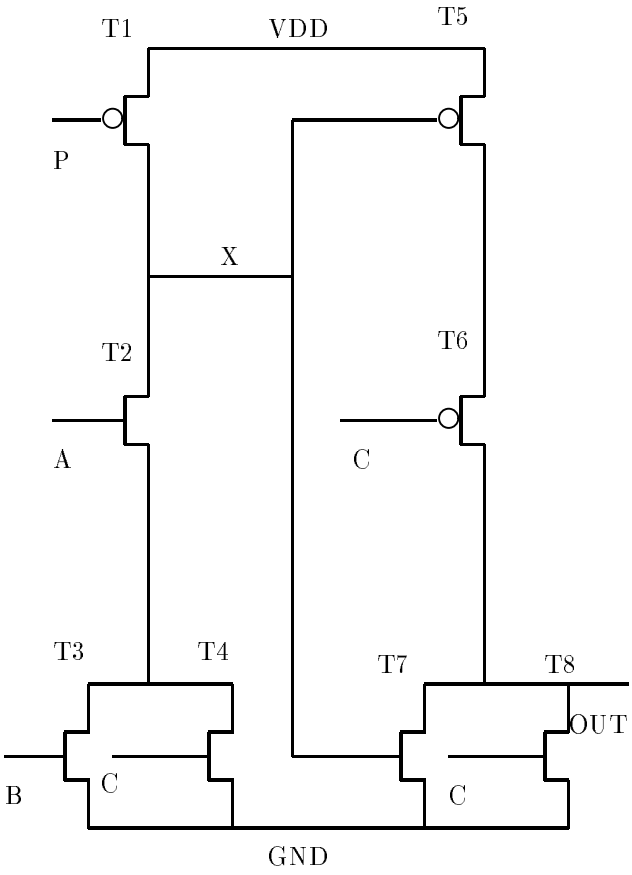

Figure 3. The circuit.

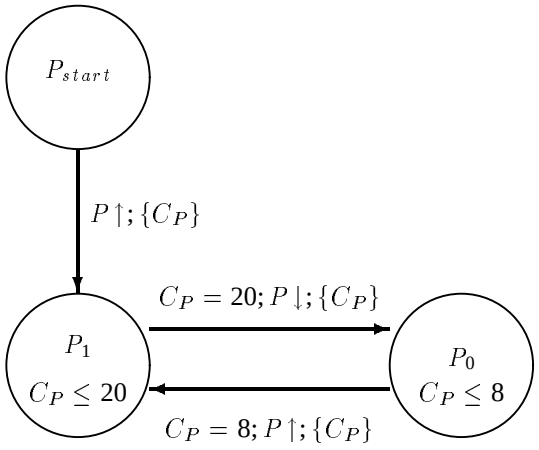

Figure 4. The automata for the input signal P.

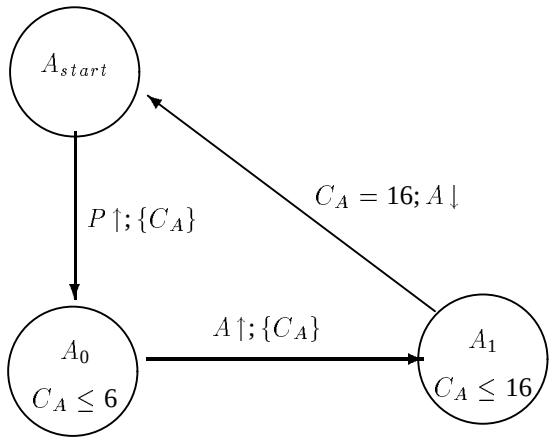

Figure 5. The automata for the input signal A.

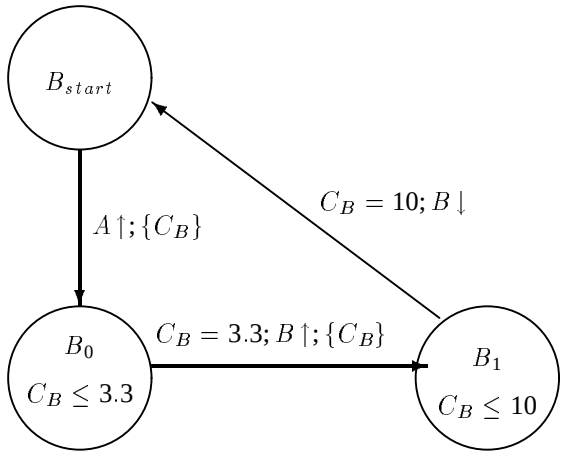

Figure 6. The automata for the input signal B.

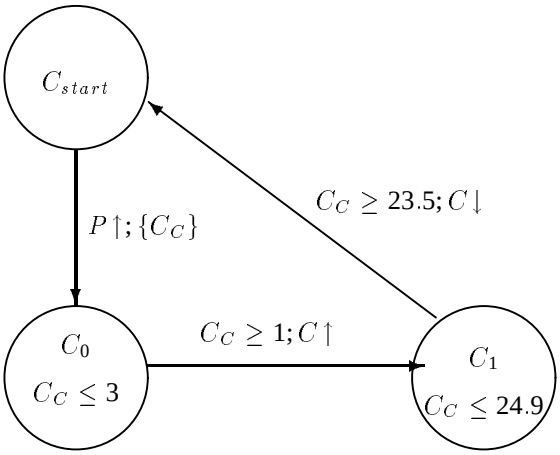

Figure 7. The automata for the input signal C.

As a first step we compose the automata of P, A, B, and C with T_1, T_2, T_3, and T_4. This gives us an automaton with 193 states, 529 transitions and 8 clocks. We then define a formula X as

$$(T_1 = On) \wedge \neg((T_2 = On) \wedge ((T_3 = On) \vee (T_4 = On)))$$

and apply a special utility that adds the event labels $X\uparrow$ and $X\downarrow$ to all the transitions of the product that change the value of X. Then we compose the automaton with the remaining transistors T_5, T_6, T_7 and T_8, and obtain the whole circuit as a timed automaton with 769 states, 2683 transitions and 12 clocks.

The property that we want to verify is the absence of short-cuts, namely current flowing from VDD to GND for all the possible executions starting at the initial state. This is expressed in the formula

$$\neg((T_1 = On) \wedge (T_2 = On) \wedge ((T_3 = On) \vee (T_4 = On))) \wedge$$

$$\neg((T_5 = On) \wedge (T_6 = On) \wedge ((T_3 = On) \vee (T_7 = On)))$$

We call KRONOS to evaluate this formula against the system and the result is false. In this case we obtain an evaluation file containing a generalized counter-example (appendix 2), which indicates classes of timed event sequences that invalidate the formula. From this output one can extract a concrete counter example and deduce which change of parameters will guarantee the satisfaction of the formula. For

example, in this case a bad sequence of events is:

$$(P\uparrow, 0) \rightarrow (T_5\uparrow, 1.0) \rightarrow (C\uparrow, 1.5) \rightarrow (T_4\uparrow, 2.5) \rightarrow$$
$$(T_8\uparrow, 2.5) \rightarrow (A\uparrow, 5.0) \rightarrow (T_2\uparrow, 6.0) \rightarrow$$
$$(B\uparrow, 8.3) \rightarrow (T_3\uparrow, 9.3) \rightarrow (B\downarrow, 18.3) \rightarrow$$
$$(T_3\downarrow, 18.3) \rightarrow (P\downarrow, 20.0) \rightarrow (T_1\uparrow, 21.0)$$

If, on the other hand, we change the system such that P must wait more than 21 time units before falling, we can make sure that $A\downarrow$ and $T_2\downarrow$ occur before $T_1\uparrow$ and, indeed, when we verify this property with the modified system we get a positive answer from KRONOS. The verification CPU time (on Sun SparcStation 20 with 64MB of memory) was 5 seconds when the property was false and 27 seconds when it was true.

References

[1] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[2] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, Symbolic Model-Checking: 10^{20} States and Beyond, *Proc. LICS'90*, Philadelphia, 1990.

[4] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Proc. FORTE'94*, pages 227–242, Bern, Switzerland, October 1994.

[5] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. 1995 IEEE RTSS'95*, Pisa, Italy, December 1995. IEEE Computer Society Press.

[6] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, The tool KRONOS. In *Workshop on Hybrid Systems and Autonomous Control*, DIMACS, New Jersey, October 1995.

[7] D. Dill, Timing Assumptions and Verification of Finite-State Concurrent Systems, in J. Sifakis (Ed.), *Automatic Verification Methods for Finite State Systems*, LNCS 407, Springer, 1989.

[8] E. Clarke and E.A. Emerson, *Design and synthesis of synchronization skeletons using branching-time temporal logic*, In *Workshop on Login of Programs, LNCS 131*, 1981.

[9] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[10] K.L. McMillan, *Symbolic Model-Checking: an Approach to the State-Explosion problem*, Kluwer, 1993.

[11] O. Maler and A. Pnueli, Timing Analysis of Asynchronous Circuits using Timed Automata, in P.E. Camurati, H. Eveking (Eds.), *Proc. CHARME'95*, 189-205, LNCS 987, Springer, 1995.

Appendix 1

This is the source code for defining the input signal A and its transistor T_2. Note that all constant are multiplied by ten.

```
/* this is the generator of the input
signal A. It can rise between  0 and 6
time units after start (i.e. after P
goes up) and it falls 16 units later */

#states 3
#trans 3
#clocks 1
CA

state: 0
prop: A_start /* before P starts  */
invar: true
trans:
true  => P_UP; reset{CA}; goto 1

state: 1
prop: A_0
invar: CA <= 60
trans:
true  => A_UP; reset{CA}; goto 2

state: 2
prop: A_1
invar: CA<=160
trans:
CA=160 => A_DOWN; reset{}; goto 0

/* The Transistor T1 enabled by A
It rises 1 time unit after A_UP and
falls immediately after A_DOWN */

#states 3
#trans 3
#clocks 1
C2

state: 0
prop: T2_0
invar: true
trans:
true => A_UP; reset{C2}; goto 1

state: 1
prop: T2_rising
invar: C2<=10
trans:
C2=10 =>  T2_UP; reset{C2}; goto 2

state: 2
prop: T2_1
invar: true
trans:
true => A_DOWN; reset{}; goto 0
```

This is the source code for the property we wish to verify:

```
init impl ab (
  (not (T1_1 and T2_1 and (T3_1  or T4_1)))
  and
  (not (T5_1 and T6_1 and (T8_1  or T7_1)))
)
```

Appendix 2

This is the beginning and the end of the counter-example information provided by KRONOS when a the verification of a property fails. It is a sequence of generalized states and transitions. A generalized state consists of a location and a set of constraints on the system clocks, including an additional variable T representing the global time. A generalized transition is an event that can take place when certain conditions on the clock are satisfied. From these constraints one can deduce a family of system runs (one of which is described in the body of the paper) which invalidate the property.

```
[0, CP=0 and C1=0 and CA=0 and C2=0 and CB=0
and C3=0 and CC=0 and C4=0 and C5=0 and C6=0
and C7=0 and C8=0 and T=0]

CP=0 => P_UP X_DOWN ;
        RESET{ CP CA CC C5 } ;
        goto 1

[1, C5<=10 and CP=C1 and CP=CA and CP=C2 and
CP=CB and CP=C3 and CP=CC and CP=C4 and
CP=C5 and CP=C6 and CP=C7 and CP=C8 and CP=T
and 0<=T<=10]

CP<=200 and CA<=60 and CC<=30 and C5=10
     => T5_UP ;
        RESET{ C5 } ;
        goto 4

[4, 10<=CP and CC<=30 and CP=C1 and CP=CA
and CP=C2 and CP=CB and CP=C3 and CP=CC and
CP=C4 and CP=C5+10 and CP=C6 and CP=C7 and
CP=C8 and CP=T and 10<=T<=30]

CP<=200 and CA<=60 and 10<=CC and CC<=30
     => C_UP ;
        RESET{ C4 C8 } ;
        goto 14

[...
```

```
some states and transitions deleted
...]

[442, CP<=200 and 133<=CA and CA<=160 and
CP=C1 and CP<=CA+60 and CP=CC and CP<=C4+40
and C4+20<=CP and CP=C5+10 and CP=C6 and
CP=C7 and CA=C2+10 and CA=CB+133 and
CA=C3+43 and CA<=C4 and C4=C8 and and CP=T
and 153<=T<=200]

CP=200 and CA<=160 and CC<=249
     => P_DOWN ;
        RESET{ CP C1 } ;
        goto 474

[474, C1<=10 and CA<=160 and CP=C1 and
CP+140<=CA and CP+200=CC and CP+160<=C4
and C4<=CP+180 and CP+190=C5 and CP+200=C6
and CP+200=C7 and CA=C2+10 and CA=CB+133
and CA=C3+43 and CA<=C8 and C4=C8 and
T=200]

CP<=80 and C1=10 and CA<=160 and CC<=249
     => T1_UP ;
        RESET{} ;
        goto 516

[516, 10<=CP and CA<=160 and CP=C1 and
CP+140<=CA and CP+200=CC and CP+160<=C4 and
C4<=CP+180 and CP+190=C5 and CP+200=C6 and
CP+200=C7 and CA=C2+10 and CA=CB+133 and
CA=C3+43 and CA+10<=C8 and C4=C8 and
T=CP+200 and 210<=T]
```