Extended Timing Diagrams as a Specification Language*

Stefan Lenk
Lehrstuhl Rechnerstrukturen
Universitaet Passau
D - 94030 Passau

Abstract

Hardware designs increasingly evolve to distributed systems composed of multiple interacting components working in parallel. Extended Timing Diagrams presented in this paper are an intelligible graphical specification language especially suited for the description of such communicating systems. The formal semantics of extended timing diagrams is defined in terms of a process calculus. This formalization permits applying the language as an entrance to formal design systems.

1 Introduction

Hardware systems are more and more designed as distributed systems with extensive interactions between the different components. This communication is mainly characterized by strict temporal constraints. Specifications of those often asynchronous systems using formalisms like temporal logic or process calculi cause problems in applications due to their high complexity. Obvious approaches for a graphical specification of interacting systems are described in [4, 14].

Timing diagrams perfectly meet the requirements for a clear graphical specification of the I/O-behaviour of an interacting system. In [3] timing diagrams are used as specifications for controller synthesis using special event-graphs as a semantical basis. Also targeting on automatic synthesis [6] uses timing diagrams as illustrative background for signal-transition graph specifications. With a similar graph semantics Khordoc et al. [5] define hierarchical timing diagrams used for the generation of simulation stimuli for modules described in VHDL. However, the proposed graph-based semantical models of timing diagrams are not sufficiently abstract to support different design styles. Moreover, these approaches cannot handle data paths. "Formalized timing diagrams" in [2] tackle this problem by integrating a hardware description language into timing diagrams.

In [11] abstract timing diagrams restricted to qualitative timing are used for the formal verification of hardware designs. The formal semantics of timing diagrams is given by a translation into temporal logic.

The specification language "extended timing diagrams" presented in this paper combines the advantages of a clear and comprehensible graphical representation of temporal aspects with the algorithmical description of data manipulations in terms of a hardware description language. This duality considerably increases the expressiveness and applicability of extended timing diagrams in contrast to conventional timing diagrams.

2 Motivation

Asynchronous communication among interacting systems is mainly characterized by causal temporal relations, often tight absolute timing requirements on events and to some extent by the exchanged data values. Timing diagrams excellently visualize this *input-output behaviour* of concurrent communicating systems. They allow illustratively representing different *temporal constraints* and they naturally describe *asynchronity*. Every timing diagram implicitly specifies possible *parallelism* to its highest degree. It always represents the finest possible granularity of parallelism thus retaining most freedom for a variety of possible subsequent implementations. The widespread use of timing diagrams for the description of interface circuits gives evidence for their practical applicability. Moreover, interacting systems commonly comprise, often minor, data paths executing certain data manipulations such as the adaptation of data formats or the recognition of data errors and so on. Conventional timing diagrams do not adequately support a specification of data paths. Commonly, if at all, they are given as textual remarks. Extended timing diagrams presented here permit extenting graphically specified events by *data annotations*. These annotations al-

*This work is supported by the European Community ESPRIT-project No. 6128 FORMAT.

gorithmically describe data operations related to an event. The language of extended timing diagrams is based on a strict *compositional approach*. Every extended timing diagram can be arbitrarily composed of a set of basic (graphical) primitives. However, evidently a single extended timing diagram is not sufficient for the specification of a complex system behaviour. Therefore a *structuring concept* for specifications is provided. Again, this concept permits arbitrarily *composing* a specification from single extended timing diagrams. This compositional approach eases a clear specification process and eg. allows a stepwise construction of a specification by iteratively adding new constraints.

The *formalization* of a specification in terms of the formal specification language T–LOTOS enables their rigorous analysis. This can be the *validation* of specific system properties eg. by testing or simulation, or the extensive *verification* of a system in all details. A formalization also facilitates the application of *formal synthesis methods* which automatically or interactively produce a guaranteed correct implementation of a specification.

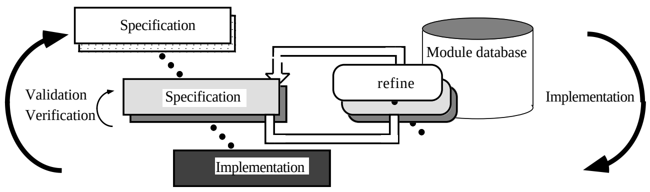

Figure 1: Formal synthesis

In our project, synthesis is mainly organized as an interactive *bottom-up* design process (figure 1)[12, 13]. Hardware modules are iteratively selected by a designer in order to *refine* (or "implement") a given specification. After every module selection the refined specification is analyzed whether it still *"satifies"* the original specification. All formal methods used there work on *transitional systems*, a semantical model of the formal specification language T–LOTOS.

Chapter 3 defines the language primitives and the structuring concept of extended timing diagrams. Chapter 4 describes the formal semantics of a specification. In chapter 5 an example specification is presented and chapter 6 gives a conclusion.

3 Language Definition

In the following the graphical appearance of the different extended timing diagram constituents will be briefly presented together with an additional informal explanation of their semantics.

Definition : *Extended Timing Diagram*
An *extended timing diagram* consists of
⊳ a **local design interface** IF_{TD}. It comprises

- the finite set P_{TD} of *names* of the observable *ports* considered within the timing diagram.
- the *direction modes* of the ports within the timing diagram (input/output port).
- the *(data-)types* of the values communicated on each port

⊳ a finite **waveform set** A_{TD} consisting of a single waveform A^p_{TD} for every port $p \in P_{TD}$. Each waveform is constituted by

- an *event sequence* ES_p for every port $p \in P_{TD}$
- *data annotations* related to the events in ES_p

Every action in A_{TD} therefore is related to a port in P_{TD}, i.e $PORT_{TD} : A_{TD} \rightarrow P_{TD}$.
⊳ a finite set C_{TD} of **temporal constraints**.

Four types of constraints are distinguished:

1. *Weak constraint* $WC_n : (A^n_{TD} \times A_{TD}), n > 0$
   For every $WC_n((a_1, \ldots, a_n) \times t)$, $a_1, \ldots, a_n, t \in A_{TD}$:
   - $PORT(a_i) \neq PORT(a_j), i \neq j,\ i, j \in 1, \ldots, n$
   - $\forall\ i :\ PORT(a_i) \neq PORT(t), i \in 1, \ldots, n$
2. *Strong constraint* $SC_n : (A^n_{TD} \times A_{TD}), n > 0$
   For every $SC_n((a_1, \ldots, a_n) \times t),\ a_1, \ldots, a_n, t \in A_{TD}$:
   - $\forall\ i :\ PORT(a_i) \neq PORT(t), i \in 1, \ldots, n$
3. *Quantitative constraint* QC : $(A_{TD} \times\ A_{TD})$ or $(A_{TD} \times\ P_{TD})$ or $(P_{TD} \times\ A_{TD})$
4. *Weak initial constraint* $WCI_n : (P^n_{TD} \times A_{TD}), n > 0$
   For every $WCI_n((p_1, \ldots, p_n) \times t)$, $p_1, \ldots, p_n \in P_{TD}$, $t \in A_{TD}$: $\forall\ p_i :\ PORT(t) \neq p_i,\ i \in 1, \ldots, n$ $\diamond$

• **Local Design Interface** A single extended timing diagram specifies some aspects of the behaviour of a system by describing communication actions observable on some of the system's *ports*. Properties of the ports valid within this timing diagram, like their *direction modes* and their *datatypes* are defined in the *local design interface*.

• **Waveforms** Substantial graphical constituents of timing diagrams are *waveforms*, i.e. *sequences of communication actions* that are associated with each port. With each communication action a particular date of the declared datatype is either received or transmitted. Every waveform consists of an *event sequence*, graphically expressed by a sequence of *edges* and associated *data annotations*, given as a *textual annotation* at every edge. Data annotations are expressions in terms of an annotation language (an algorithmical subset of the VHDL–syntax). Their semantics is defined in terms of an algebraic specification language. The edges constituting an input waveform may be annotated with *variable declarations*. A data value received with such an action, is assigned to the annotated variable. A variable declaration can be supplemented with a *predicate*. It prescribes that only data communications satisfying this predicate shall be admitted. Thus a predicate restricts the data values that can be received with a particular communication action to a specific data value or a subset of the (appropriate) data domain. The edges of an output port may be annotated with *value declarations*. These are computation rules for the data values the system has to send with particular output actions. Value declarations are structured expressions that may refer to variables declared at the edges of input waveforms within the considered timing diagram.

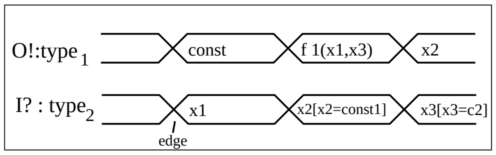

Figure 2: Graphical notation of a waveform

• **Temporal Constraints** *Qualitative constraints* (WC,SC,WCI) allow expressing a temporal ordering on communication actions from different waveforms of a timing diagram. *Quantitative constraints* (QC) specify minimum and/or maximum times to pass between two actions from arbitrary waveforms of a timing diagram. Let now be A_{TD} the waveform set of a timing diagram and P_{TD} the names of ports this timing diagram covers.

- weak constraint $WC_n : (A^n_{TD} \times A_{TD})$

A $WC = ((s_1, \dots, s_n), t)$, $s_1, \dots, s_n, t \in A_{TD}$ is graphically specified by a *weak constraint arc*, relating a set of edges corresponding to $s_1, \dots, s_n, t$ in the timing diagram. It specifies that the action t **may** (arbitrarily often) occur, if (1) presently all 'conditional' source actions $(s_1, \dots, s_n)$ occurred (2) these actions are still 'valid', i.e. neither of these actions were superscribed by a later action on the same port.

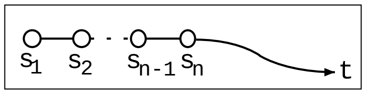

Figure 3: Notation of WC: *weak constraint arc*

Note that the names $s_1, \dots, s_n, t$ in figure 3 are merely illustrative representatives for corresponding edges of a waveform. Actually they are not part of a weak constraint arc. The similar holds for the subsequent illustrations of primitives.

- weak initial constraint $WCI_n : (P^n_{TD} \times A_{TD})$

A $WCI_n = ((p_1, \dots, p_n), t)$, $p_1, \dots, p_n \in P_{TD}$, $t \in A_{TD}$ is graphically specified by a *weak initial constraint arc*, relating a set of ports $p_1, \dots, p_n$ and an edge corresponding to t. It specifies that the action t **may** (arbitrarily often) occur, if every port $p_1, \dots, p_n$ is in its 'initial state'. Intuitivelly this means that on each of these ports previously a communication action had to occur which led to this initial state. The graphical notation resembles the notation for the weak constraint shown before.

- Strong constraint $SC : (A^n_{TD} \times A_{TD})$

A $SC = ((s_1, \dots, s_n), t)$, $s_1, \dots, s_n, t \in A_{TD}$ is graphically specified by n *strong constraint arcs*, each uniquely relating one of the edges corresponding to the source actions $s_1, \dots, s_n$ and the edge corresponding to the target action t. It specifies that (1) each of the 'trigger' source actions s_i has to occur once (in arbitrary order) before the triggered target action t can occur once and (2) a repeated occurrence of one of the 'trigger' source actions s_i is only possible after the occurrence of the previous triggered target action t.

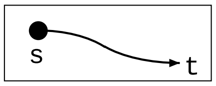

Figure 4: Notation of SC: *strong constraint arc*

- quantit. timing constraint $QC : (A_{TD} \times A_{TD})$

A $QC = (s, t)$, $s, t \in A_{TD}$ is graphically specified by a *quantitative constraint arc* relating corresponding edges in a timing diagram including a *min/max time interval* $[t_1, t_2]$. It specifies that the communication action t has to occur within the time interval $[t_1, t_2]$ after the occurrence of communication action s.

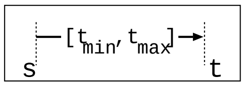

Figure 5: Notation of QC: *quantitative constraint arc*

• **Structuring** The specification of the I/O–behaviour of a complex communicating system by means of a single extended timing diagram may turn out to be very complex. A single extended timing diagram eg. does not allow specifying different alternative situations and very large extended timing diagrams soon forfeit their clearness. Therefore a concept for the *structurization* of timing diagram specifications is introduced.

Definition : *Complex Extended Timing Diagram*

A *complex extended timing diagram* CTD consists of

▷ a **design interface** IFS_{CTD}, comprising

- the set P_{CTD} of the *portnames* , used within the complex timing diagram.
- the set PDS_{CTD} of *direction mode sets* and
- the set PTS_{CTD} of *datatype sets*

for each port in P_{CTD} and each timing diagram instance in TDI_{CTD} .

▷ a finite set TDI_{CTD} of **instances** of extended timing diagrams.

▷ a **set** A_{CTD} **of waveform sets** A^p_{CTD} for each port $p \in P_{CTD}$, i.e. $A_{CTD} := \bigcup_{p \in P_{CTD}} A^p_{CTD}$. A_{CTD} contains a set of mutual exclusive waveforms for every port.

$\triangleright$ a set C_{CTD} of **temporal constraints** on the actions in A_{CTD}, each related to a particular timing diagram instance. $\diamond$

A (well-formed) complex timing diagram can be constructed from a set of extended timing diagrams by arbitrarily combining *instances* of these timing diagrams using the *alternative* and the *sequential timing diagram operator*. Note that the notion of instantiation enables a *multiple* use of an extended timing diagram within a specification. The *alternative* operator $\oplus$ permits specifying alternative situations using different extended timing diagrams. The *sequential* operator $\triangleright$ allows splitting up a large extended timing diagram into several smaller ones.

An *extended timing diagram specification* then consists of 1. the definition of the *system design interface*. It determines the ports of the system, their direction modes and the types of data that can be communicated on the ports. 2. a finite set of *extended timing diagrams* whose local design interfaces have to be convenient with the system design interface. 3. a *complex timing diagram* in terms of a timing diagram expression combining instances of extended timing diagrams from 2. using the timing diagram operators $\oplus$ and $\triangleright$.

4 Formal Semantics of a Specification

The formal semantics of an extended timing diagram specification is given in terms of the formal specification language T–LOTOS. T–LOTOS is based on the specification language LOTOS which is founded on process calculi, mainly CCS [9], extended by a data communication concept. T(IMED)–LOTOS is an extension of LOTOS, enriching the pure temporal ordering of actions in LOTOS by a notion of quantitative time [10].

• **T–LOTOS** T–LOTOS describes a system behaviour by a hierarchy of *processes*. A process is defined by a *behaviour expression* which is either a combination of subprocesses or, at bottom-level, a sequence of atomic *actions*. For a detailed description of LOTOS or T-LOTOS resp. see [1][10].

Let B, B_1, B_2 be variables denoting "behaviour expressions", t, t_1, t_2 time variables of a time domain, T a time interval $[t_1, t_2]$, $a1, \ldots, a_n$ action names taken from a universe of action names, A a set of such action names, typ, typ_1, typ_2 datatypes from a set of dataypes, $v, v_1, \ldots, v_m$ variables of some predefined datatypes, $E, E_1, \ldots, E_m$ data expressions denoting data values of those datatypes. A subset of the available operators and their T–LOTOS syntax are given Table 1.

Operator	syntax
Stop	*stop*
Action Prefix	$a\{t\}?v : typ; B$
	$a\{t\}!E; B$
Timed Choice	$a\{t \; in \; T\}?v : typ; B$
	$a\{t \; in \; T\}!E; B$
Choice	$B_1 [] B_2$
Parallelism	$B_1 \vert[A]\vert B_2$
Process Definition	$P[a_1, .., a_n](v_1 : typ_1, ., v_m : typ_2) := B$
Proc Instantiation	$P[a_1, .., a_n](E_1, \ldots, E_m)$

Table 1: Subset of T–LOTOS syntax

Informally the depicted language elements are interpreted in the following way: *stop* is the completely inactive behaviour. '$a\{t\}?v : typ; B$' or '$a\{t\}!E; B$' resp., is a behaviour that is willing to engage in an action named a at a certain instance of time t, receiving *?* a data value of type *typ* or sending *!* a data value determined by E. The subsequent behaviour is B. '$a\{t \; in \; T\}?v : typ; B$' or '$a\{t \; in \; T\}!E; B$' describes a similar behaviour but the participation in action a may happen at any instance of time within the time interval T. '$B_1 [] B_2$' is the mutual exclusive choice between behaviour B_1 and behaviour B_2. '$B_1 |[A]| B_2$' is the parallel composition of the behaviours B_1 and B_2. They "synchronize" via the actions in the set A (i.e. they have to engage in these actions in common). Wrt. all other actions B_1 and B_2 may act independantly. A process definition '$P[a_1, .., a_n](v_1 : typ_1, \ldots, v_m : typ_2) := B$' defines a formal process P that behaves like B engaging in the (formal) actions named $a_1, .., a_n$ using formal data parameters $v_1, ..., v_m$. A process instantiation '$P[a_1, .., a_n](E_1, \ldots, E_m)$' is a process that behaves like the (formally defined) process P replacing the formal action names by the actual names $a_1, .., a_n$ and formal data parameters by the actual data values determined by $E_1, \ldots, E_m$.

• **Translation** The semantics of an extended timing diagram specification is defined as a recursice T–LOTOS process. This process results from a *parallel composition* of the process translations for *each graphical constituent* of the extended timing diagram instances used in the specification and the process equivalents of the *timing diagram operators* combining these timing diagrams. This compositional approach is described for a single basic timing diagram in [12] and extended to single data–annotated timing diagrams in [7]. Given an extended timing diagram specification its semantics is a process

$ETDS[Act] \; := EventSeq[Act_1] \; |[S_1]| \; Data[Act_3] \; (init \; values) \; |[S_2]| \; Constraints[Act_2] \; [S_3]| \; TDops[Act_4])$

where *EventSeq* is the parallel composition of the process translations of all event sequences.

Constraints is the parallel composition of the process translations of all constraints. *Data* is the process translation of the data annotations. It controls the reception and the transmission of data values. *initvalues* are the the initial values of the variables declared there. *TDops* is the parallel composition of the process translations of the timing diagram operators. $Act, Act_i, i = 1, \ldots, 4$ are the sets of action names the corresponding processes participate in and $S_i, i = 1, \ldots, 4$ are appropriate sets of actions the parallelly composed processes must engage in together.

In the following the T–LOTOS process translations of two basic graphical primitives (as defined in 3) of extended timing diagrams are presented. Let now be $CTD = \{IFS, A, C, TDI\}$ a complex timing diagram, $IFS = \{P, PDS, PTS\}$ its local design interface and $TDI = \{td_1, \ldots, td_n\}$ the instances of extended timing diagrams constructing CTD.

- Translation of **event sequences**.

Let $A^p := \bigcup_{td \in TDI} A^{p_{td}}$ be the waveform set of mutual exclusive waveforms $A^{p_{td}}$ related to port p defined in different timing diagram instances td. Then $A^{p_{td}}_{CTD} := \{a^{td}_1, \ldots, a^{td}_{l_{td}}\}$ is the totally ordered set of actions (waveform) defined in the timing diagram instance td at port p, assuming $a^{td}_1 \prec \ldots \prec a^{td}_{l_{td}}$. The semantics of the mutual exclusive event sequences of a waveform set A^p on a particular port p defined in timing diagram instances $td_1, \ldots, td_n$ is given as a T-LOTOS process P_{A^p}:

```
$P_{A^p}[a^{td_1}_1, \ldots, a^{td_n}_{l_{td_n}}] :=$
$a^{td_1}_1 ?x : typ_m; \ldots ; a^{td_1}_{l_{td_1}} ?x : typ_n; P_{A^p}[a^{td_1}_1, \ldots, a^{td_n}_{l_{td_n}}]$
   $[] \ldots []$          /*waveform td1 OR..OR waveform tdn*/
$a^{td_n}_1 ?x : typ_o; \ldots ; a^{td_n}_{l_{td_n}} ?x : typ_p; P_{A^p}[a^{td_1}_1, \ldots, a^{td_n}_{l_{td_n}}]$
```

where typ_m,typ_n,typ_o,typ_p are appropriate types.

Note that this process does *not* restrict the data values communicated with the actions it participates in. It solely prescribes the temporal ordering of these actions. The process *EventSeq* representing *all* event sequences in a specification then results from the parallel composition of the process translations of the event sequences for every system port:

$$EventSeq[Act] := \underset{p \in P}{|[]|} \; P_{A^p}[a^{td_1}_1, \ldots, a^{td_n}_{l_{td_n}}]$$

- Translation of **quantitative constraints** C^{QC}

Every quantitative constraint $qc = (s^{td}, t^{td})$ with associated time interval $[t_1, t_2]$ defined in a timing diagram instance td is translated to a process P_{qc}:

```
$P_{qc}[s^{td}, t^{td}] :=$
 $s^{td} ?x : typ_s; P1_{qc}[s^{td}, t^{td}]$  $[]$  $t^{td} ?x : typ_t; P_{qc}[s^{td}, t^{td}]$
$P1_{qc}[s^{td}, t^{td}] :=$
    $t^{td}\{t \; in[t_1, t_2]\} ?x : typ_t; P_{qc}[s^{td}, t^{td}]$
   $[]$    /*t in [t1,t2] OR again s in (0,t2)*/
 $s^{td}\{t \; in(0, t_2)\} ?x : typ_s; P1_{qc}[s^{td}, t^{td}]$
```

P_{qc} specifies that after an action s^{td} an action t^{td} has to occur within the time interval $[t_1, t_2]$. Further actions s^{td} may occur up to t_2, i.e. within the time interval $(0, t_2)$, leaving some time for the action t^{td} to occur ('(,),[,]' denote open or closed interval bounds).

5 Example : Traffic Light Controller

As an example specification the *traffic light controller* [8] is presented. An intersection of a major highway and a minor farmroad is controlled by traffic lights. A detector loop signals the presence of cars on the farmroad. Three major situations characterize the traffic at this intersection:

[**CarDetected**] If the highway lights are green they will stay green for at least a fixed period of time (t_{long}). If a car is detected on the farmroad the highway lights will turn to yellow after their minimal green period (provided the detected car does not turn round before). If the farmroad lights are green [**TimeOut**] then if permanently cars are detected on the farmroad the farmroad lights will turn to yellow after a fixed period of time (t_{long}). But if there was no more car we would also be satisfied if (sluggish) lights would manage to turn yellow after their maximal green period, *or alternatively* [**NoCar**] if no more cars are detected on the farmroad the lights will be able turn to yellow at once.

It is reasonable that the detection of a car [CarDetected] always precedes the alternatives of (1) an expiration of the maximal green period on the farmroad [TimeOut] or (2) the premature absence of cars on the farmroad [NoCar]. Together with the knowledge of the commonly strict ordering green, yellow, red, aso. of traffic light colours and the consideration of some security aspects one can easily find the extended timing diagram specification of this traffic light controller as depicted in figure 6. An idea of the T–LOTOS spec-

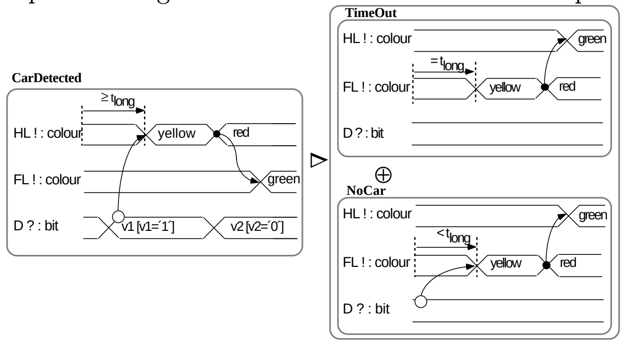

Figure 6: Timing Diagram Specification of the TLC

ification obtained from the extended timing diagram specification in figure 6 is given in figure 7.

The transitional system semantics of this T–LOTOS

```
SPECIFICATION TLC[DET1_CarDetected,DET2_CarDetected,HL1_Car...
                                    ...,FL3_TimeOut,FL3_NoCar]:NOEXIT
BEHAVIOUR
((( EventSeq[DET1_CarDetected,..]    |[DET1_CarDetected,..]|
Data[DET1_CarDetected,..](0,0))      |[DET1_CarDetected,..]|
Constraints[DET1_CarDetected,...] ) |[HL1_CarDetected,..]|
TDops[HL1_CarDetected,...,FL2_NoCar])
WHERE
PROCESS EventSeq[DET1_CarDetected,DET2_C...,FL3_NoCar]:NOEXIT:=
AltSeq[DET1_CarDetected,DET2_CarDetected] |[]|
AltSeq[HL1_CarDetected,HL2_CarDetected]    |[]|
AltSeq[FL2_TimeOut,FL2_NoCar,FL3_TimeOut,FL3_NoCar] |[.]| ....
ENDPROC
PROCESS Data[DET1_CarDetected,...](v1:BIT,v2:BIT):NOEXIT:=
HL1_CarDetected!yellow;Data[DET1_CarDetected,...](v1,v2)
  []...[]
DET1_CarDetected?x:BIT[x=1];Data[DET1_CarDetected,...](x,v2)
ENDPROC
PROCESS Constraints[DET1_CarDetected,...,FL3_NoCar]:NOEXIT:=
SC[HL2_CarDetected,FL1_CarDetected]|[]|WC[Det1_C..,HL1_C..]...
```

Figure 7: Excerpt of T-Lotos specification for TLC

description abstracting each communication action to an action on its related port is depicted in figure 8. It is computed from the T–Lotos specification dissolving the parallel composition operators and could be used for a subsequent synthesis.

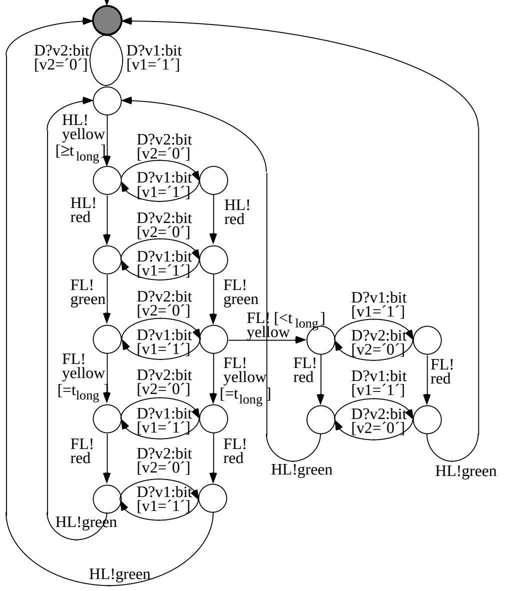

Figure 8: Transitional system semantics of TLC

6 Conclusion

A graphical specification language with comprehensible language elements and well-founded structuring concept has been presented. A sufficient modelling power is ensured by the capability to describe data paths. The formal semantics of the specification language has been given in terms of the specification language T–Lotos. A prototype for the translation of extended timing diagram specifications has been implemented. Some resulting specifications has been tested with a method to translate T–Lotos specifications to VHDL. This transformation is developed within the ESPRIT–project FORMAT[13]. The obtained VHDL-descriptions eg. could then be processed by usual high-level synthesis systems or used for simulation purposes. The main future research will concentrate on a notion of *hierarchy* of timing diagram specifications. Formally based transformations of the data paths in a timing diagram specification can be done on a combined control-data flow graph representing data annotations. For this purpose a formal semantics for such graphs in terms of process algebra will be given.

References

[1] T. Bolognesi; E. Brinksma: *Introduction to the ISO specification language LOTOS* in: R.H.J. van Eijk, C.A. Vissers, M. Diaz (ed): *The Formal description Technique LOTOS* Elsevier Science Publishers, North Holland, 1989.

[2] G. Boriello: *Formalized timing diagrams*, in: *Proceedings, The European Conference on Design Automation*, pages 372–377, Brussels, Belgium, March 1992

[3] G. Boriello; R. H. Katz: *Synthesizing Transducers from Interface Specifications*, *Proc. Int. Conf. on VLSI*, North Holland, 1988.

[4] D. Harel: *StateCharts – A Visual Formalism for Complex Systems*, Science of Computer Programming 8 (1987), 231–274.

[5] P. K. Khordoc, M. Dufresne, E. Czerny: *A Stimulus/Response System based on Hierarchical Timing Diagrams*, Publication #770, Technical report, Universitè de Montreal, 1991

[6] L. Lavagno; A. Sangiovanni-Vincentelli: *Algorithms for Synthesis and Testing of Asynchronous Circuits* , Kluwer Academic Publishers (1993).

[7] S. Lenk: *Requirements for the Graphical Specification Language 'Timing Diagrams', ESPRIT-Project FORMAT.* Technical report, University of Passau, 1992

[8] C. Mead, L. Conway: *Introduction to VLSI Systems*, Addison-Wesley 1980

[9] R. Milner: *Communication and Concurrency* Prentice-Hall (1989).

[10] J. Quemada, A. Azcorra, D. de Frutos: *A Timed Calculus for LOTOS*. In S. Vuong, editor, *Formal Description Techniques*, Vancouver, Canada, December 1989, FORTE 89.

[11] R. Schloer: *Specification and Verification of System-level Hardware designs using Timing Diagrams* EDAC'93 (1993).

[12] W. D. Tiedemann: *An Approach to Multi-paradigm Controller Synthesis from Timing Diagram Specifications*; *Proc. 1st EURO-DAC '92* (1992), 522–527

[13] W. D. Tiedemann; S. Lenk; C. Grobe; W. Grass: *Introducing Structure into Behavioural Specifications obtained from Timing Diagram Specifications*; *Microprocessing and Microprogramming 38* North Holland(1993), 581-588

[14] F. Vahid; S. Narayan; D. Gajski: *SpecCharts – A Language for System Level Synthesis Proc. 10th Int. Symp. on Comp. Hardware Description Languages and their Applications*, CHDL'91 (1991), 145–154