# COMPUTER HARDWARE DESCRIPTION LANGUAGES AND THEIR APPLICATIONS

edited by
m. breuer & r. hartenstein

# *COMPUTER HARDWARE DESCRIPTION LANGUAGES AND THEIR APPLICATIONS*

Proceedings of the IFIP TC-10 Fifth International Conference on Computer Hardware Description Languages and Their Applications
Kaiserslautern, F.R.G., 7-9 September, 1981

edited by

M. BREUER
*University of Southern California*
*U.S.A.*

R. HARTENSTEIN
*University of Kaiserslautern*
*F.R.G.*

50

1931 N·H 1981
·P·C

NORTH-HOLLAND PUBLISHING COMPANY
AMSTERDAM • NEW YORK • OXFORD

ISBN: 0 444 86279 X

*Published by:*
NORTH-HOLLAND PUBLISHING COMPANY – AMSTERDAM • NEW YORK • OXFORD

*Sole distributors for the U.S.A. and Canada*
ELSEVIER NORTH-HOLLAND, INC.
52 Vanderbilt Avenue
New York, N.Y. 10017

**Library of Congress Cataloging in Publication Data**

IFIP TC-10 International Conference on Computer
Hardware Description Languages and Their Ap-
plications (5th : 1981 : Kaiserslautern,
Germany)
Computer hardware description languages and their
applications.

Includes index.
1. Electronic digital computers--Design and
construction--Data processing--Congresses.
I. Breuer, Melvin A. II. Hartenstein, Reiner.
III. IFIP TC-10. IV. Title.
TK7888.3.I27 1981 621.3819'582'028542 81-16873
ISBN 0-444-86279-X AACR2

PRINTED IN THE NETHERLANDS

## PREFACE

This volume is the final version of the proceedings of "CHDL '81", which also includes invited papers and other presentations not printed in the preprints. CHDL '81 means: the 5th International Symposium on Computer Hardware Description Languages and their Applications, held September 1981 at Kaiserslautern University, F.R.G. This Symposium is the fifth one in a series of symposia having been held in New Brunswick (New Jersey), Darmstadt, F.R.G., New York and Palo Alto (California). All these preceding meetings have been organized as ACM/IEEE events, whereas this fifth one was organized by IFIP WG 10.2 in cooperation with ACM, IEEE Computer Society, the European Research Office, the NTG (Nachrichtentechnische Gesellschaft), the GI (Gesellschaft fuer Informatik), and the Commission of the European Economic Community.

We appreciate the valuable publicity support provided by these organizations. We also gratefully acknowledge the financial support by the European Research Office (London), CTM Computertechnik Müller G.m.b.H. (Konstanz), Honeywell Bull A.G., (Köln), Nixdorf Computer (Paderborn), Kienzle Apparate G.m.b.H.(Villingen), and Digital Equipment G.m.b.H. (München). This list does not include financial support contributed later than June 1981 when this preface was written.

The most importance part of the work was done by Professor Melvin Breuer (University of Southern California), who served as the Program Chairman. We gratefully acknowledge his excellent work in solliciting papers, organizing the reviewing process, and setting up the program. It should be mentioned that the very careful selection of papers (at least three referees per paper), and a large number of submissions helped to assemble a program of very high quality. We also appreciate the services of the members of the Program Committee: J. Duley (Vice-Chairman), M. Barbacci (USA), D. Borrione (F), H. de Man (B), H.W. Lawson jr. (S), D. Lewin (GB), A. Parker (USA), R. Piloty (D), and W. vanCleemput (USA).

Of course without the contributions of all the submitting authors from all over the world it would not have been possible to assemble a really international program of such high quality. We appreciate all their efforts. We also appreciate the efforts spent in the preparation of papers which were not accepted because of the large number of high quality submissions.

Especially valuable have been the efforts of D. Borrione, publicity chairperson for Europe, and of W. Sherwood, publicity chairman for the rest of the world. Let me especially acknowledge the layout of the call for papers, the advance program, the program booklet, and also part of this volume, which was provided by W. Sherwood. The work of the Finance Chairman, Dr. Egon Hörbst, and of the Local Organization Chairman, Professor Dieter Maass, is also gratefully acknowledged. We should be aware of the fact that the result of their difficult and very important work was visible temporarily only since it is neither published nor distributed.

Last, but not least, Professor Robert Piloty and Dr. Mario Barbacci should be mentioned. Professor Piloty, as Chairman of IFIP Technical Committee TC-10, and Dr. Mario Barbacci, as Chairman of IFIP Working Group WG 10.2, brought this conference series to the attention of IFIP. It was due to their efforts that the organization of this series of symposia was conveyed to IFIP.

Reiner W. Hartenstein
General Chairman IFIP CHDL '81

## A MESSAGE FROM THE PROGRAM CHAIRMAN

Professor Melvin A. Breuer
University of Southern California
Los Angeles, California 90007

Welcome to the Fifth International Conference on Computer Hardware Description Languages and their Applications. I believe this year's conference will be particularly exciting and informative, and am sure that these proceedings will be a valuable contribution to the technical literature. From a review of the submitted papers, it is clear that the study of CHDLs is entering a new era of maturity. Though a significant number of submitted papers dealt with traditional subjects, such as describing new CHDLs, many papers were concerned with formal aspects of CHDLs and new application areas. Some of these topics cover in depth analysis of some attributes of CHDLs, such as formal definitions of syntax and semantics and the impact of these properties on verification, as well as formal studies on behavior and structures.

In Table 1 we indicate the number of papers submitted and accepted, broken down by country.

TABLE 1: DISTRIBUTION OF PAPERS

| Country | Number Submitted | Number Accepted |
|---|---|---|
| Australia | 2 | 1 |
| Bulgaria | 1 | 0 |
| Canada | 1 | 1 |
| England | 2 | 2 |
| France | 4 | 2 |
| Germany | 8 | 3 |
| Hungary | 3 | 1 |
| India | 2 | 0 |
| Italy | 1 | 1 |
| Japan | 2 | 2 |
| The Netherlands | 1 | 0 |
| New Zealand | 1 | 0 |
| Poland | 3 | 1 |
| Spain | 1 | 0 |
| Sweden | 1 | 0 |
| Switzerland | 1 | 1 |
| USA | 11 | 6 |
|  | 45 | 21 |

In addition to the submitted papers, the program committee solicited several outstanding researchers to present invited surveys and/or tutorials. Since a written record of these presentations does not appear in these proceedings, I would like to acknowledge these people here. Their contribution to the quality of this conference is substantial, and is greatly appreciated.

*SYNTAX AND SEMANTIC REQUIREMENTS OF CHDLs*

*M. Barbacci*
Carnegie-Mellon University, Pittsburgh, PA USA

*VERIFICATION OF CHDL DESCRIPTIONS*

*S. Crocker*
USC Information Sciences Institute, Marina del Rey, CA USA

*AUTOMATED LOGIC SYNTHESIS*

*J. Darringer*
IBM-Thomas J. Watson Research Center, Yorktown Heights, NY USA

*EXPERIENCES IN THE USE OF CHDLs FOR CUSTOM DESIGN OF CMOS INTEGRATED CIRCUITS*

*U. Hedengran*
The Royal Institute of Technology, Stockholm, S

*DESCRIPTION LANGUAGE-BASED INTEGRATED CAD SYSTEM FOR ELECTRONIC AND LOGICAL DESIGN*

*J. Mermet*
IMAG, Grenoble, F

*HIGH-LEVEL LANGUAGE AND PROGRAMMING ENVIRONMENT REQUIREMENTS FOR INTEGRATED VLSI CAD SYSTEMS*

*A.C. Parker*
University of Southern California, Los Angeles, CA USA

*CONLAN: A METALANGUAGE FOR CHDLs*

*R. Piloty*
Technische Hochschule Darmstadt, Darmstadt, D

*MODELING AND STYLE — A USER'S VIEWPOINT*

*W. Sherwood*
Digital Equipment Corporation, Hudson, MA USA

*CAD TOOLS FOR THE SYNTHESIS OF HARDWARE AND SOFTWARE*

*G. Zimmerman*
Honeywell, Incorporated, Bloomington, MN USA

It is my pleasure to acknowledge the tremendous help I have received from my colleagues in processing the 45 papers received, almost all of which were sent out for three reviews. First, I would like to thank Professor Alice Parker who helped in almost all aspects of the review process. Secondly, I would like to thank Jim Duley, Alice Parker and Reiner Hartenstein who met with me on April 14 to design the program. Finally, I would like to thank and acknowledge the contributions of the reviewers without whose effort a fair analysis of the papers could not have been made.

R. Alali
W. Anderson
P. Kakowski
M. Barbacci
D. Bixler
D. Borrione
S. Budkowski
P. Camposano
S. Ching
F. Christley
Y. Chu
J. Clary
B. Cohen
T. Cooper
W. Cory
S. Crocker
J. Crowley
J. Darringer
S. Dasgputa
D. Dietmeyer
J. Djordjevich
H. Dortmans
J. Duley
C. Durante
G. Estrin
I. Eveking
P. Foulk
R. Frankel
S. Fuller
M. Gordon
R. Goodell
J. Grason
R. Hartenstein
E. Hassler
F. Hautin
J. Hayes
G. Hellstrand
F. Hill
E. Horbst
R. Ibbett
W. Johnson
R. Klar
P. Kornerup
H. Lawson
V. Lesser
C. Leung
D. Lewin
G. Lipovski
R. Little
H. Loomis
P. Losleban
H. de Man
E. Marino
P. Marwedel
M. Medina
P. Meinen
J. Mercier
J. Mermet
R. Mueller
G. Musgrave
M, McFarland
R. McGuffin
A. Nagle
D. Nash
Y. Ohno
A. Ohsawa
B. Overman
A. Parker
D. Patel
R. Piloty
S. Rao
R. Razouk
C. Rose
W. Rosenstiel
S. Sastry
W. Sherwood
G. Shiva
B. Shriver
D. Siewiorek
J. Silvester
A. Singh
E. Snow
S. Su
M. Tokoro
J. Tracey
E. Ulrich
W. van Cleemput
S. Vegdahl
M. Vernon
E. Villar
C. Visser
M. Yoeli
S. Zaky
G. Zimmerman

## SEMANTICS AND VERIFICATION

*COMPUTER HARDWARE DESCRIPTION LANGUAGES AND THEIR APPLICATIONS: M. Breuer, R. Hartenstein (eds)*
***North-Holland** Publishing Company*
*© IFIP, 1981*

# THE WORKER MODEL OF EVALUATION FOR COMPUTER HARDWARE DESCRIPTION LANGUAGES

Dominique Borrione
Laboratoire IMAG
B. P. 53X
38Ø41 Grenoble Cedex - FRANCE

This paper presents an operational model to define the semantics of CHDL's. This model is expressed in terms of a hierarchy of workers who communicate by exchanging messages. The use of the model to specify concurrency, parameter passing policies and time behavior is demonstrated. CONLAN function, activity and description segments are taken as examples.

## I - INTRODUCTION

In recent years, considerable attention has been given to the formal definition of programming language semantics; a number of methods (operational [LiZ.75], axiomatic [Hoa.69], denotational [Ten.76]) can now be applied to express the semantics of sequential programs [Liv.78].

In the meantime, while many new Computer Hardware Description languages (CHDL) were being published [Com.74, Lip.77], few of them, if any, had their constructs explained by a precise model: natural language sentences, at best diagrams, continue to prevail. One of the few attempts to give a formal specification of hardware [FrS.79] seems to be applicable only at a very abstract and early stage of architecture design. Nevertheless, if one considers the many levels of detail (gate network, register transfer, microprogram, ...up to complex system architecture) for which CHDL's have been defined, the variety and complexity of language constructs far outgrow those of programming language constructs. As a simple example, programming languages have a single concept of "variable", whereas CHDL's have a broader notion of "carrier", split into terminals, clocks, registers of various kinds etc..., all with different time, loading and memory properties.

A method for defining unambiguously the meaning of CHDL primitives for a broad range of description levels is indeed a necessity, both for the implementors of such languages, and for the hardware designers who use them. And the methods developped for programming languages, even those proposed for non-deterministic and concurrent processes [Kah.79] fail to be applicable to the semantic definition of CHDL's, or become too complex and awkward, when timing, elementary parallelism and memory properties have to be expressed.

It is one of the objectives of the CONLAN Working Group to remedy this deficiency of existing CHDL's, by proposing a formal method for the definition of CHDL's at various levels of details [PBB.80d]. This method is based on the concept of language derivation, pictured on figure 1. A new language LLK is derived from an already defined language LI by the definition of new TYPE, FUNCTION, ACTIVITY and DESCRIPTION segments in terms of those known in LI. This mechanism ensures that, starting with BCL, all the languages of the CONLAN family are formally defined.

However, this semantic definition would be incomplete if the implied lowest level PSCL, and the various categories of segments, were not themselves defined in terms of a precise semantic model. It is the purpose of this paper to present the salient features of this model of computation.

The Worker Model of Evaluation has been inspired by the Actor Model of Hewitt [HeB.77a-b, Hew.77]. Although specifically designed for CONLAN, this model is so general that it can be used to represent the behavior of any discrete description language, not necessarily defined as a CONLAN family member. In the following however, we shall give as an illustration of the model the evaluation of selected CONLAN primitives. The reader is referred to [PBB.80a-c] for a presentation of PSCL, BCL, of FUNCTION, ACTIVITY and DESCRIPTION segments, and of the model of time in terms of intervals and steps.

## II - PRESENTATION OF THE WORKER MODEL

The evaluation of a description is compared to a task to be performed in a mill. A number of workers are present, in fact there are as many as needed. There is one boss who is responsible for starting the evaluation and getting the result. Workers may be hired and fired. Workers may be busy or waiting for something to do. A worker starts working when another worker asks him to do so, and he always responds to the one who asked him for the task. Communication between workers is done by questions and answers, which will be called messages.

Workers are highly specialized; some can do only integer addition, others can do only division. Workers responsible for complex tasks ask other workers specialized in appropriate subtasks for their assistance. The global task is thus performed by a hierarchical team.

Workers do not know all the other workers in the team. A worker knows those workers who command him, and those that he commands.

No worker can do more than one task at a time. If two or more commands are simultaneously directed to the same worker, that worker randomly selects and responds to each of them in sequence. We are not interested in the physical time which is necessary for messages to arrive to their destination, nor for tasks to be performed. We do assume however that this physical time is finite, and that every message gets serviced in finite time after its arrival. The only point of interest is the before-after relationship between all the messages. A message is sent before a task is performed before a response is returned.

A task of evaluation is associated with every primitive statement, every segment invocation . A worker is associated with each task. Moreover, there is a worker for each carrier declared and each instance of description used. Each worker has a unique name.

### 1 - Context

Every worker has a special sign called "context". The context is either his own name, or the name of a worker placed higher in the hierarchy. Only a restricted category of workers has the privilege to give their name as context. These workers then always have their own name as context: in CONLAN, they correspond to instances of descriptions. The boss of the team is such a worker.

A worker gets his context when he is hired, and keeps it thereafter, according to the following rule:

Let A and B be two workers such that A commands B. If B is not his own context, then B has the same context as A.

A worker always writes his context on his requests; this information is used by the receiver to verify his access rights, or to hire workers with the same context. The context, in the worker model, serves to describe scope rules.

## 2 - Messages

There is a fixed number of categories of messages. Each category of messages is directed to, or emitted by, workers attached to a given category of segments. For instance, a COMPUTE message is always directed to a worker attached to a function, and a RESULT message is always emitted by a worker attached to a function when he returns the computed function result.

A message is therefore constituted of a keyword, defining its category, possibly followed by one or more parameters; a message includes the name of the worker who has sent it, and his context when appropriate.

A message can either be readily understood by the worker who receives it, or be too complex and require the cooperation of other workers, to evaluate it. Parameters that are immediately understood are constant denotations, and declared object names. Parameters that require evaluation are expressions.

## 3 - General principles of evaluation

The hardware designer who describes a digital system defines a DESCRIPTION segment. Compiling this segment corresponds, in our model, to the creation of a team of workers for the task of evaluating the outer DESCRIPTION segment. The workers of the team represent the semantics of all segments and carriers in the DESCRIPTION.

The total number of workers varies during evaluation. Compile time actions correspond to hiring all workers who will be present during the whole evaluation process, and establishing their hierarchical links. In CONLAN one "permanent" worker is hired for each instance of DESCRIPTION used, for each invocation of a STATIC FUNCTION or ACTIVITY segment, and for each one of their local carriers. The workers who are dynamically hired and fired correspond to non static operation invocations, and their local carriers.

Messages are being exchanged when the hardware designer asks for an evaluation of his description. Two special workers, outside the hierarchical team, pre-exist:

The "environment" worker serves as interface between the hardware designer and the team, interprets the evaluation commands, and acts as communication supervisor. The environment is responsible for sending the first message to the team boss to start the evaluation.

The "recruiter" is capable of providing new workers when needed, and is directed all hiring requests.

Upon receipt of his initiating message, the team boss
- sends to the recruiter a set of messages to hire all necessary temporary workers.
- when all hired workers are received, sends to each worker associated to a subtask an initiating message.

The process is recursively repeated, down the hierarchy of workers, until workers associated to elementary tasks have been initiated.

A worker responsible for an elementary task performs his task immediately, and responds to the one who commanded him. A worker responsible for a complex task, once all subtasks have been initiated, waits until all answers have been received. If at least one answer is an error message, an error is answered to the one who commanded the task. Otherwise, after additional checks when necessary, the worker fires all temporary workers that he commanded, and gives a positive answer to the one who commanded the task. Answer messages are transmitted back along the hierarchy, up to the team boss.

Definition

A step of evaluation is the set of all messages being exchanged between the initiating message that the environment sends to the team boss, and the response of the team boss to the environment.

After one step of evaluation, depending upon the semantics of the language in which the description has been written, and stability conditions which are required or not, a new step is initiated, or this is the end of the evaluation.

Rule

A step of evaluation has no error iff it contains no error message.

An equivalent statement is:

A step of evaluation has no error iff the team boss has received no error message for this step.

## III - MESSAGES AND MESSAGE SKELETONS

### 1 - Messages

A message is a 4-tuple in which the second and fourth components may be empty:

(worker_name, context, keyword, parameter_list)

where worker_name and context are respectively the name and the context of the sender of the message. We shall use an informal notation to write messages:

worker_name: context, keyword(parameter_list)

The context part is not necessary on all messages, and in the following we shall indicate it only when its presence is required:

worker_name: keyword (parameter_list)

If the parameter list is empty, we abbreviate further:

worker_name: keyword

Let "w" be a worker, and "c" his context. Messages exchanged during one step of evaluation belong to one of the following categories:

- w: c, EVAL(p1, p2, ... pn)
  directs a segment worker to evaluate his actual parameters

- w: EVALUATED
  answer of a segment worker that has successfully evaluated all actual parameters.

- w: c, START
  directs a segment worker to perform the task described by the segment body

- w: DONE
  answer of a worker who has completed his evaluation task without error

- w: c, COMPUTE(p1, p2, ... pi)
  directs a function worker to evaluate his actual parameters, his body (if any) and his return expression

- w: RESULT(r)
  answer of a function worker, where "r" is the result of the computation

- w: ERROR(explanation)
  answer of a worker who could not successfully fulfil his task. "explanation" is a string, which explains the reason of the failure. "explanation" may be empty, or a very sophisticated sentence, depending upon the implementation.

In messages EVAL and COMPUTE, the pi are the actual parameters of the operation. An actual parameter can be :
- a constant
- a worker name (corresponding to a declared object)
- an expression, written in prefix notation

Messages are also exchanged between operation segments and the recruiter, to
- hire a worker for an internal operation (nested invocations)
- get a fresh carrier (local declarations of carriers in operation segments)

Such messages take the following form :

- w: c, SEND(operation_designator)
  request to the recruiter to provide a worker for the operation specified

- w: c, SEND(type_designator)
  request to the recruiter to provide a carrier worker of the type specified

- recruiter: RECEIVE(worker_name)
  answer of the recruiter, which provides the requested worker.

Other messages, with keywords TYPE, EMPTY, GET, SET, PUT are requests exclusively addressed to carrier workers, and will be explained in the appropriate section.

In the following, to shorten the text, we shall write "function plus", or "plus", instead of "the worker attached to function plus". Further, the transmission of messages will be denoted with an arrow, with the receiver of the message identified at the point.

```
w1: message -> w2    is to be interpreted as w2 receives "message" from w1.

w1 <- w2: message    is to be interpreted as w1 receives "message" from w2.
```

The second form is preferred for a returned object, or an acknowledgement.

To represent a set of messages, we shall make a distinction between concurrent and sequential exchanges. The following format indicates concurrent messages:

```
w1 : message1 -> wr1
w2 : message2 -> wr2
w3 : message3 -> wr3

wn : messagen -> wrn
```

whereas sequential messages are indicated by:

```
w1 : message1 -> wr1
   +
w2 : message2 -> wr2
   +
w3 : message3 -> wr3
   +

   +
wn : messagen -> wrn
```

## 2 - Message skeletons

A worker interacts with other workers by accepting and sending messages. Depending upon his task, a worker is capable of accepting some messages, and not others. The description of acceptable messages, and of their appropriate answer, is done in terms of "message skeletons".

A message skeleton is a 4-tuple, with the same components as a message, and written in a similar format, in which one or several components except the keyword is a formal element. A formal element is an identifier in which the first character is "$".

Two formal elements are pre-defined:
- $me designates the worker himself
- $mine designates his own context

When a worker receives a message, he compares it with the skeletons of acceptable messages. Recognition is first attempted using the keyword. If a match is found, and all non-formal components of the skeleton are identical to the corresponding ones in the received message, then the formal components of the skeleton are evaluated. Formal components in first and second position are immediately replaced. Formal elements in the parameter list are typed; the actual parameters in the received message may require hiring of function workers for their evaluation, and the results must belong to the types specified .

If no skeleton can match a received message, the message is refused, and an error message is answered to the sender.

Examples of message skeletons

```
$a: $c, EVAL ($pl:int, $p2:bool)
$2: EVALUATED
recruiter: RECEIVE($007)
$me: $mine, SEND(int.+)           int.+ designates plus on integers
```

Rules

The scope of a formal element is limited to the description of the task in which it appears.

In a task description, all the occurrences of the same formal element stand for the same worker name or the same object. When a formal element is identified, all the occurences of that formal name in the task description are replaced with the object or the worker name that it represents, for the duration of the task being performed.

## 3- Description of message emission and reception

The specification of an acceptable message, in a task description, is written:

```
        ** message-skeleton
```

where "**" is read as "receive".

The specification of a message to be sent, in a task description, is written:

```
        message-skeleton ==> receiver
    or  receiver <== message-skeleton
```

where "<==" or "==>" is read as "send", and "receiver" is either a formal name or the recruiter.

Message receptions and emissions can be indicated as un-ordered, or sequential, grouped or individually, using the same writing conventions as the format that we adopted in paragraph 1 for actual messages arriving to their actual receiver.

Example

The following text describes a task of multiplication by 2 on integers. Lines are numbered for convenience.

```
(1)  ** $1: $2, COMPUTE ($p:int)
           +
(2)  $me: $mine, SEND(int.*)  ==> recruiter
           +
(3)  ** recruiter: RECEIVE($mult)
           +
(4)  $me: $mine, COMPUTE($p, 2) ==> $mult
           +
(5)  ** $mult: RESULT($twotimesp)
           +
(6)  $1 <== $me: RESULT(twotimesp)
```

This script specifies the task as 6 sequential exchanges of messages. It reads as follows:

(1) Receive from $1, with context $2, a COMPUTE message with one integer parameter $p.
(2) Hire a worker for integer multiplication.
(3) Receive this worker, called $mult.
(4) Send to $mult a COMPUTE message with parameters $p and 2.
(5) Receive from $mult the result of the multiplication.
(6) Send this result to $1.

Let "a" be a worker, "ca" the context of "a", "times2" the name of a worker hired for the above task, and "m" a multiplication worker. An execution of the multiplication of 10 by 2 results in the following exchanges of actual messages:

```
a: ca, COMPUTE(10) -> times2
     +
times2: ca, SEND(int.*) -> recruiter
     +
times2 <- recruiter: RECEIVE(m)
     +
times2: ca, COMPUTE(10,2) -> m
     +
times2 <- m: RESULT(20)
     +
a <- times2: RESULT(20)
```

## 4 - Conditional forms

As in PLASMA, we adopt 2 conditional forms, written in a syntax close to that of a case statement. The first one is used to test a formal name; the second one is used to test a message. In both cases, the result of the test selects a sub-task among several alternatives.

A formal name is tested with:

```
SELECT formal-name
   (value1: subtask1)
   (value2: subtask2)
```