In "Tool integration and design environments" edited by F.J.Rammig, North-Holland to appear in 1988.

Open Framework of Interactive and Communicating CAD tools.*

L.Claesen, R.Severyns, P.Six, W.De Rammelaere, H.De Man†

IMEC Kapeldreef 75, B-3030 Leuven Belgium

J.Cockx, Ph.Reynaert

Silvar-Lisco, Abdijstraat 34, B-3030 Leuven, Belgium

G.Schrooten

Philips Res. Labs., Prof. Holstlaan, NL-5600 JA Eindhoven, Netherlands.

Abstract

This paper describes an open system architecture for *interactive and communicating* CAD programs. This system is concentrated around the concept of the SPI structure procedural interface between CAD tools. The goal of the system architecture is to provide a direct *interaction* and *feedback* between the primary design tools (schematics editors, symbolic layout editors, module generators etc.) and intelligent verification tools (electrical debugging, timing verification simulation etc.). The tools run in *parallel* and have *bidirectional interactive communication* by pointing to objects in primary design inputs and passing information to verification tools and by allowing verification tools to highlight objects in primary design editors.

Keywords: CAD, Open frameworks, CAD architecture, Object Oriented Systems, tool integration.

1 Introduction

The design of VLSI chips or VLSI modules encompasses several aspects where CAD tools can assist. Even today most designs are directly entered in layout editors. However several, more appropriate design entry tools become accepted such as schematics editors [22] for semi-custom designs, symbolic layout [14,15] and parameterized module generators [17,16].

During the global design process, the verification and characterization of the correctness of the circuits takes a considerable amount of the design time. For the verification of MOS circuits logic-, switch-level- and circuit level simulation is used most often. Currently new approaches are being developed for verification to reduce the requirements for extensive simulations. For the correctness verification of MOS circuits, knowledge based approaches are used [7] for verifying design soundness about composition rules, electrical rules, electrical debugging. Other tools have been developed for the verification of timing correctness [8,9,10,11]. The major goal of these analysis tools is to try to reduce the need for extensive circuit simulations. Circuit

*Research performed within the scope of the ESPRIT-1058 project

†Professor at Kath. Univ. Leuven Belgium

simulation will however always remain important. Therefore circuit simulation techniques for large circuit modules are being developed based on waveform relaxation techniques [12] and explicit integration techniques [13].

One bottleneck in the efficient application of these verification techniques is the interactivity between verification and simulation tools and the designer.

The current design practice is that CAD tools are run *one at a time* and that *communication of information is via files and cross-reference lists.* This is extremely time consuming in a verification or debugging phase where the feedback between design definition and design debugging is currently taking most of the designers time. An other disadvantage of the current CAD tools is that they often require different formats for representing netlist information, which necessitates the use of cross reference lists and makes it harder for the designer to relate information from a simulator or a timing verifier to the original information.

To allow for a much faster feedback between knowledge based verification tools and the module designer, an open and interactive system architecture is under development as described in this paper.

2 Open system architecture for interactive CAD.

Much attention is currently put on the issue of open system architectures for electronic CAD programs [1] and promising prototypes are under development [2,3,4,5]. Up to the authors knowledge, there are currently no *open* CAD environments commercially available that allow to *embed* strongly interactive CAD tools in an easy way.

This paper explains the outline for an *open architecture* allowing for *bidirectional* communication between interactive CAD tools running in parallel [18]. It is conceived in such a way that it is possible to interactively communicate between user-interface tools and simulation and verification tools by user-pointing and screen highlighting of structural objects. The architecture is set up in such a way that individual tools can be developed independently. (or that independently developed tools can be integrated with minor work). The system architecture majorly integrates around the concept of structural data that is communicated between tools. To integrate existing tools easily, they are not much constrained. Tools are allowed to have their own "primary data". It is only required that the tools are also able to generate the structure information in a standard way from this "primary data". Because there are not too many constraints on the tools themselves allows that existing CAD tools can be integrated without too much difficulty. This fact together with a good definition of *procedural interfaces* for communicating to other tools contributes to the aspect of an *open system architecture.*

Important in this architecture description is the philosophy of allowing CAD tools to communicate interactively to the user via the primary input descriptions. In this way a much faster feedback is achieved to the designer and also a more efficient design cycle is achieved.

In this system the tools are organized around the concepts of:

Common Database Organization . This does not imply the use of one database or the use of the same formats. (Nevertheless it is encouraged.) This is to access the files as a whole in a standard way, not the contents.

Communicating application programs: In the underlying system it is required that two or more programs can communicate with one another in a *bidirectional* way. This is necessary to allow *highlighting* of and *pointing* to structural items. To achieve this a structure communication standard SPI is used [19].

Uniformization of structure data: By structural data is assumed data about:

cells . These items are also known by the database interface.

components or instances of cells.

ports of cells

ports of components

nets

as well as some general attributes that can be associated with each of the structural entities described above. *Parameters* are a special type of attributes that are associated with cells (formal parameters) and components (actual parameters).

Databases used by application programs: To be able to put as few constraints on application programs as possible, the general rule is that *each application tool is allowed to have its own formats for databases or data files.* The requirement is only that the designer, or some design tools have to indicate to the DMS database interface which view (generated by which tool) contains the primary data for the cell from which the structural data can be derived. The structural data is standardized. This data should be extracted from the own data structures of structure generation tools and passed in a standardized way to tools that act on the structural data. Because the other databases are not that constrained is advantageous to incorporate existing CAD tools in the system. It also puts less constraints on new tools that have to be developed.

Primary Data Assumption. In the module design environment it is assumed that the designer declares for each cell which tool defines the correct primary data. (This is done through the DMS.). In the database this is reflected because for each cell there must be information which tool has defined the primary data for that cell. The primary data corresponds to the information that the designer has initially entered as specification of the design. An example is symbolic layout [14]. Secondary data is then for example the physical layout generated from the symbolic layout. Other "so called" primary data may also exist but is not considered as primary in this text. An example is a cell with the symbolic layout declared as primary data by the designer, and of which also a schematic has been made. Here the schematic data is "so called" primary data but not "the" primary data. The meaning is that the schematic is not the primary definition for the structure of the cell. The structure as represented by the schematic can be compared to the primary structure by a netlist comparison program such as for example WOMBAT [20]. The DMS design management system has to be aware of which view primarily defines the structure for each cell, because this information will be used by preference for further verification work.

Implementation details: All operations of CAD programs that are communicating together via SPI are considered to be *synchronized*. This means that only one tool will be activated at a specific time. Other processes are then in a waiting state.

3 Tool summary and interaction.

Figure 1 gives the major communication. The notational conventions used are indicated in the figure. The functionality of the system is of prime importance. An implementation of the system can be done over one or more processes running on one or more machines.[1] The DTM

[1]In the actual implementation all tools are in separate processes. For the understanding of the system architecture, one could as well think that all tools are integrated in one large program.

data & tool manager is the main controller and selects active projects (or cells) and activates CAD-tools. The DMS *design management system* selects the database files from the operating system file system. The SPI *structure procedural interface* is the *structure communication bus* between the CAD tools.

Now follows a further discussion of the functionality of the individual modules depicted in fig.1.

3.1 DTM : Data & Tool Manager.

The Data and Tool Manager is the main process controlling the global CAD activity. This tool allows to define the *current project(s)* on which actions are to be performed. Then the CAD tools such as editors, verifiers and simulators can be called. DTM is aware of the available tools and of the available cells and it controls the switch box function around the SPI procedures such that data is routed from the appropriate structure generation tools to the appropriate structure receiver tools.

The DTM consists of the main control loop with the user. The designer can:

- either indicate projects to be used by the programs.
- or control the execution of application programs and inter process communication between application programs.

The functionality of the DTM could be extended on top of the functionality of existing "monitor" programs encountered in several CAD systems [25]. The extension required by this architecture is the ability to manage several *communicating* CAD-tools, instead of only managing one tool at a time.

3.2 Application programs.

The application programs can be classified in two major groups:

1. Structure generation programs.
2. Structure receiving programs.

The two classes of application programs are further discussed in the next subsections. One program can also belong to the two classes. An example is a preprocessor program that takes structure and transforms it into other structure.

3.2.1 Structure generation programs.

The structure generation programs consist of the application programs that have a direct graphical interaction with the user. These programs are for the most part editor-like and communicate in a graphic way. These programs are mostly used to provide *primary input* of design data to the system.

Examples of these foreground application programs are:

- schematics editor.

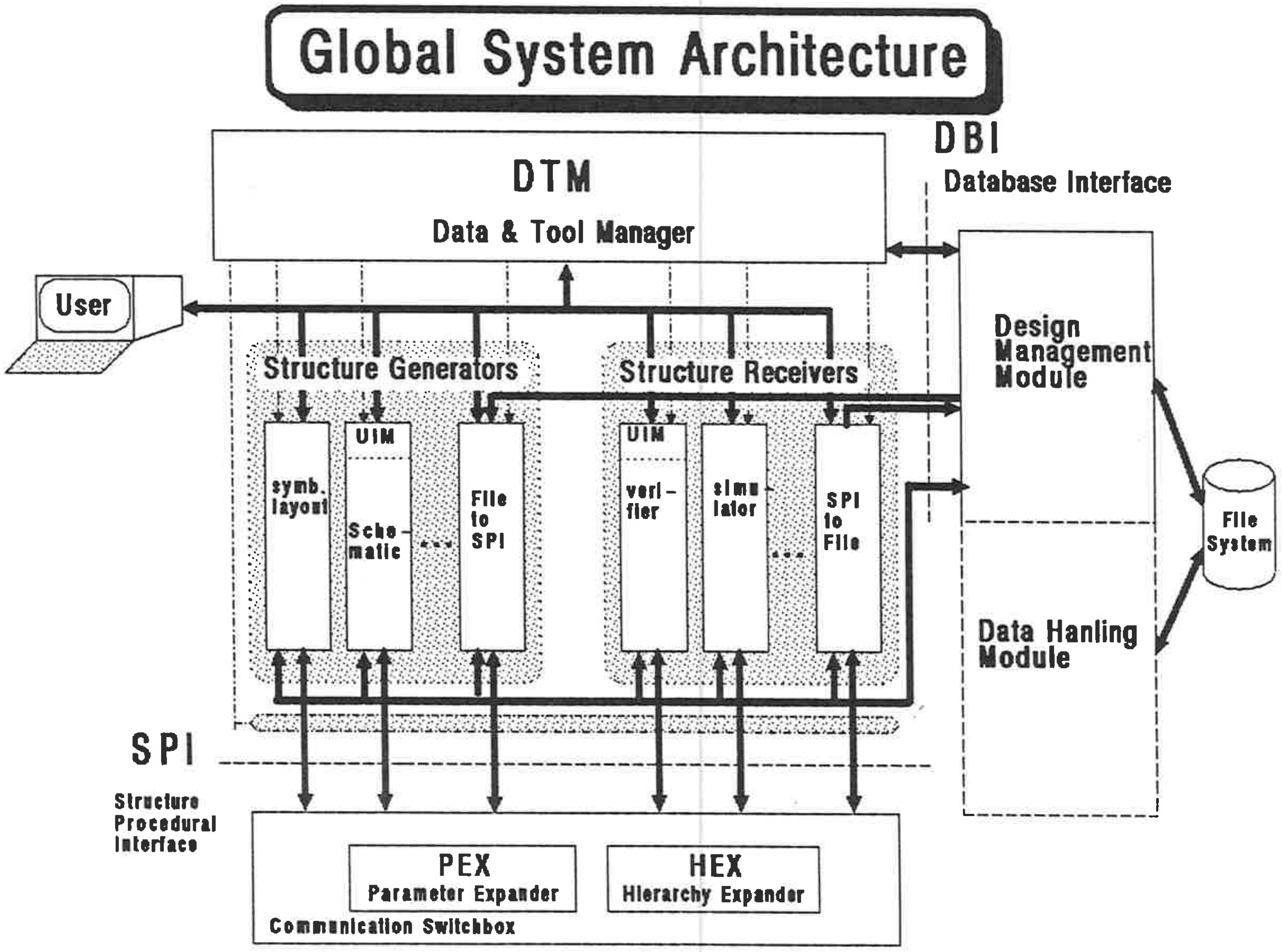


Figure 1: System Architecture

- layout editor.
- sticks editor.
- textual editor.
- algorithmic floorplanner, parametric module generator.

3.2.2 Structure receiving programs.

The structure receiving programs are the programs that do not have a direct input capability for primary design data.

Examples are:

- simulators [12,13]
- timing verification programs such as SLOCOP [10]
- electrical verification programs such as DIALOG [7].
- floorplanner, block place & route
- place & route programs.

The major reason behind this subdivision of application programs, is that it is intended that there will be a strong interaction between the application programs and the designer/user of these programs. Therefore structure generation and structure receiving programs must be interlinked closely. This allows the designer to pass information on his design from the structure generation programs to the structure receiving application programs and get feedback in the reverse way by user pointing to structural items and highlighting. In this way two tools can act as one global integrated tool.

3.3 SPI : Structure procedural interface.

The structure procedural interface [19] is a switch box and a standard way to communicate *in a bidirectional way* structure information (cells, components, ports, nets and attributes on these) between structure databases, editors (for interactive highlighting) and verification and simulation programs.

As part of the structure procedural interface there are two expanders available HEX and PEX. A hierarchy expander (HEX) which removes hierarchy in a hierarchical description and a parameter expander (PEX) which removes structural parameters in a parameterized structure description. This is necessary if verification or simulation tools, that only understand flattened information, want to communicate with one of the user-interface editors by pointing and highlighting.

SPI is a standard *procedural interface* on structural data, suited for programs. The same interface can be used to communicate between editor and files, in which all structured information is stored, and between these files and verification and simulation programs, but only in one direction (no highlight). Besides SPI, there is also a directly derived textual representation, to be used for archival purposes and for the communication with tools where no coupling via procedures is possible. Conversion between these direct textual representations in files and SPI procedures are provided by tools `File-to-SPI` and `SPI-to-File` in figure 1.

Dual to the SPI procedures there is also a designer oriented textual representation HILARICS. The coupling between the *designer oriented textual representation* and the application programs will be provided by a HILARICS to SPI compiler. In this way application tools should only use the SPI procedural interface. HILARICS is then "the" primary input.

3.4 HEX : Hierarchy expander.

The hierarchy expander provides a bidirectional link between structural data on level 1 (hierarchical) to structural data on level 0 (flat). To provide the possibility for bidirectional communication, local storage of datastructures is necessary to keep the links between structure data on level 0 and structure data on level 1. The user determines via the DTM and the application tools which data has to be expanded.

3.5 PEX : Parameter expander.

The PEX module is similar in functionality to the HEX module. It provides a bidirectional translation between tools that generate structural data on level 2 (parameterized) and structural data on level 1. The same observations as made for the HEX module also hold here. This expander is mainly used for the expansion of parameterized structure descriptions in the HILARICS language.

3.6 DMS : Design Management System..

The Design Management System performs a mapping of design objects such as *designs*, *cells*, *views* and *versions* in the file system. A design object is the basic building block to be used at the design management level (schematic, layout, behavioral model of a cell, parameterized description of layout, ...). It returns a unique identification of a file on which the other tools can act. The DMS is not aware of the contents of the design objects. The contents of the design objects is determined by the tools that act on it.

There are many alternative implementations possible for a DMS [23,24]. The specific implementation is however hidden behind the DMS-access procedures. Depending on the needs in a specific design environment, the implementation can have extended features and facilities (such as protections etc.) or can be elementary. Within the scope of the system architecture, the functionality of handling basic design objects is the major requirement.

The DMS also knows for each cell which view defines the *primary data* that determine the structural data for the view under consideration. This information with each cell is required such that the DTM together with the SPI can decide which tool has to be called for each level of cell, to generate the required structural information for the structure receiving tool.

In the DMS also an attribute with *the usage level* of each cell should be indicated. The *usage level* is for example `circuit level`. This means that the SPI procedures should call starting from the top cell, structure generating tools until the circuit level (transistors, capacitors etc.). This information could be entered for example via the DTM.

3.7 UIM User Interface Module

The UIM provides a consistent user interface to be used for all tools. It is not essential for the system architecture as such. However if two tools are being integrated, a consistent user

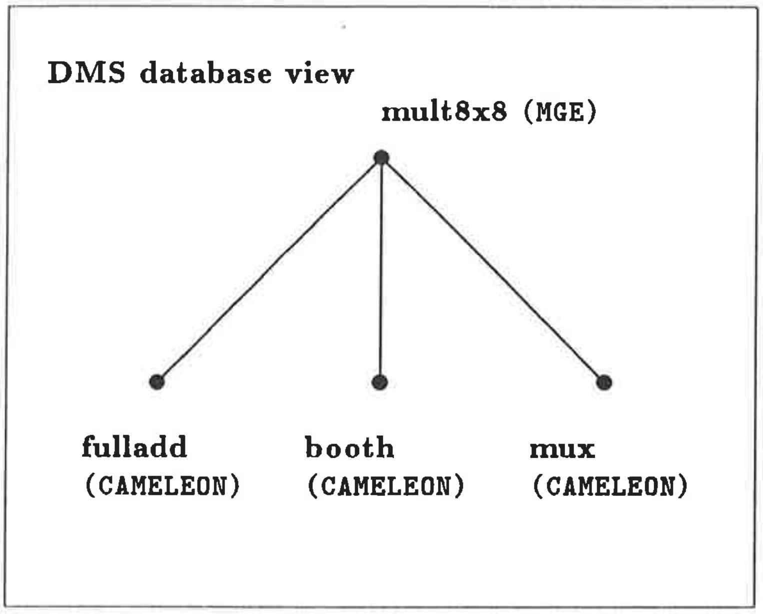


Figure 2: Sample database organization.

interface is advantageous: menus, windows, highlighting, selecting, message handling are done in a consistent way. Therefore a common UIM is encouraged because it contributes to the openness of the system architecture.

4 Description of a sample session.

To understand better the operation of the system architecture, an informal description of a sample session is given.

4.1 Sample design situation description.

Assume that a design of a multiplier has been made and that an instance of an 8 by 8 multiplier is available in the database. Assume that the cells are organized by the DMS as shown in figure 2. The 8 by 8 multiplier is represented in the DMS as the cell **mult8x8**. Assume that this cell is constructed using components from cells **fulladd**, **booth** and **mux**. The cell **mult8x8** is generated by the MGE module generation program [16]. Assume also that the designer has defined that the structure is primarily defined by MGE for the top cell **mult8x8**. The structure of the leaf cells is primarily defined by the symbolic layout program CAMELEON. Notice that the DMS is aware of which program primarily defines the structure.

Assume that the designer has indicated to the DMS the *usage levels* for the cells. In that sense the *circuit level* has been assigned to the cells **fulladd**, **booth** and **mux**. This is necessary afterwards for the SPI implementation to know until what level the design has to be passed to the application program by the SPI procedures. DIALOG [7], a circuit simulator like SPICE or SWAN [12], a timing verifier like SLOCOP [10] all require *circuit level*.

Suppose that the designer wants to do interactively an electrical verification by using the DIALOG [7] program.

Via the DTM the designer can interact with the DMS. He can see what information is in the databases.

Interacting with stand-alone tools such as editing-only sessions is done in the normal way. This means that projects from the DMS are selected by the designer through the DTM and that the appropriate CAD tools are started in a stand-alone way.

In this way the information in the database for `MGE` or `CAMELEON` could for example be created.

The case where more tools are activated at the same time is described in the following sessions.

4.2 Initiation of a verification session.

Suppose the designer is at the point of checking the **mult8x8** design for electrical bugs. Therefore he first has to enter the DTM, (via a special monitor window on the screen). There the designer can browse trough the available projects managed by the DMS. He will than choose **mult8x8**. This defines the top cell in the hierarchy. Together with one DTM the designer can start one or more CAD tools that will cooperate in *a synchronous way* under the direction of the DTM.

Now the designer can start one or more CAD tools, (each as separate processes and in separate windows).

In this design application the designer could start a window (process) with the module generator `MGE` and leave `MGE` in a waiting state controlled by the DTM. In the same way `CAMELEON` could be started in a window (process) and be left in a waiting state under control of DTM. Notice that in this example situation *two* different programs are started. These programs can be made active by the DTM to do editing or by the DTM and SPI to load a specific cell and generate structural data for the SPI interface.

Suppose that `MGE` and `CAMELEON` are both active and that the designer starts DIALOG from the DTM.

After this action DTM gives control to DIALOG for doing its initialization and for loading the information for the project at hand **mult8x8**. Therefore DIALOG gives the control back to the DTM, which will now activate the SPI module.

SPI will now recursively go trough the DMS design tree and activate each of the appropriate structure generation tools with the appropriate DMS-cell indication. Each of these tools will then for each of the cells call the appropriate SPI procedures to download the structural information to the SPI module.

This means that the SPI will start from the top cell **mult8x8** and ask the DMS for the primary view. In this case this is `MGE`. SPI activates `MGE` with cell **mult8x8**. MGE then generates the structure information via the SPI calls. MGE keeps the information of this cell in its datastructure and goes back in a wait state and gives back the control to SPI. SPI then knows via the SPI calls of `MGE` which cells are used as components of **mult8x8**. Now SPI can again activate tools to generate the structure information for the cells of which it does not have the structure information yet. This is the case for the cells **fullad, booth** and **mux**. SPI therefore goes through this list one by one and performs the following actions:

> It asks the DMS for the primary view of the cell at hand. First this will be **fulladd** with primary view `CAMELEON`. Then SPI starts CAMELEON with cell **fulladd**. CAMELEON will then ask the specific `CAMELEON` file in the database for

cell **fulladd** through the DMS. It loads the information and passes the derived structural information to SPI. CAMELEON keeps this information and goes to the wait state again. Control goes back to SPI. SPI will than take the next cell (e.g. booth and repeat the same actions).

After this CAMELEON has been activated three times by SPI with three different cells. The information of these cells is each individually kept in CAMELEON datastructures and has been passed to the SPI module via SPI procedures.

Now all information is available in the SPI module. Because DIALOG only understands flat circuit descriptions, the HEX module in the SPI block will be used. In the HEX hierarchy expander all links between the hierarchical network and the expanded circuit are kept, to allow communication afterwards.

After the expansion all information in the SPI module is passed to DIALOG via SPI procedures. After this action by SPI control is given to DTM, which forwards control to DIALOG which can start its verification work.

4.3 Interactive communication between editors and application program.

Now the case of a highlight will be described. The case of a user select operation is dual to the highlight.

Suppose that DIALOG wants to communicate a possible electrical bug to the user via the editors. Therefore DIALOG will mention in its own window a textual description of the kind of bug it has found.

DIALOG will now show to the SPI module which structural item(s) need to be highlighted in the editors.

In a highlight or a select operation SPI will pass control to the user, so that the user can choose the components in which the highlights or selects have to be done. For the case of highlight, this means that after DIALOG has passed control to SPI, SPI will give control to the user and let him/her choose among the available (and possibly affected) components that he/she wants to see. This is necessary because highlights and/or selects can affect more cells in a design. It is most often even different in components of the same cell.

The control that SPI gives to the user is by a (hierarchical) table of the components used in the design.

During these highlight or select operations the designer can look at the primary data as has been entered in CAMELEON or MGE. Remember that all cells in this project **mult8x8** are available in the data structures of these structure generation tools. Remember also that SPI keeps al the hierarchical relations between the flattened circuit and the hierarchical circuit.

After the highlight or select operation, the user can indicate to the SPI process that the highlight or select operation is finished. Now SPI can give control back to DIALOG, which can do further verifications with possible highlights or selects.

After the DIALOG verification session finishes, the control is given back to the DTM.

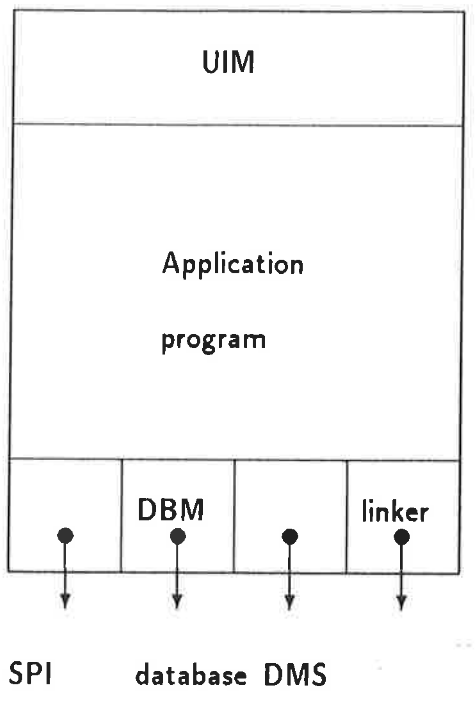


Figure 3: Integration of application program in the system architecture.

5 Interfacing of tools in System Architecture.

The interfacing of the tools in the system architecture is illustrated in fig.3. To obtain a system that is as open as possible, and also to allow a gradual integration of tools in the architecture some interfaces are mandatory and others are optional.

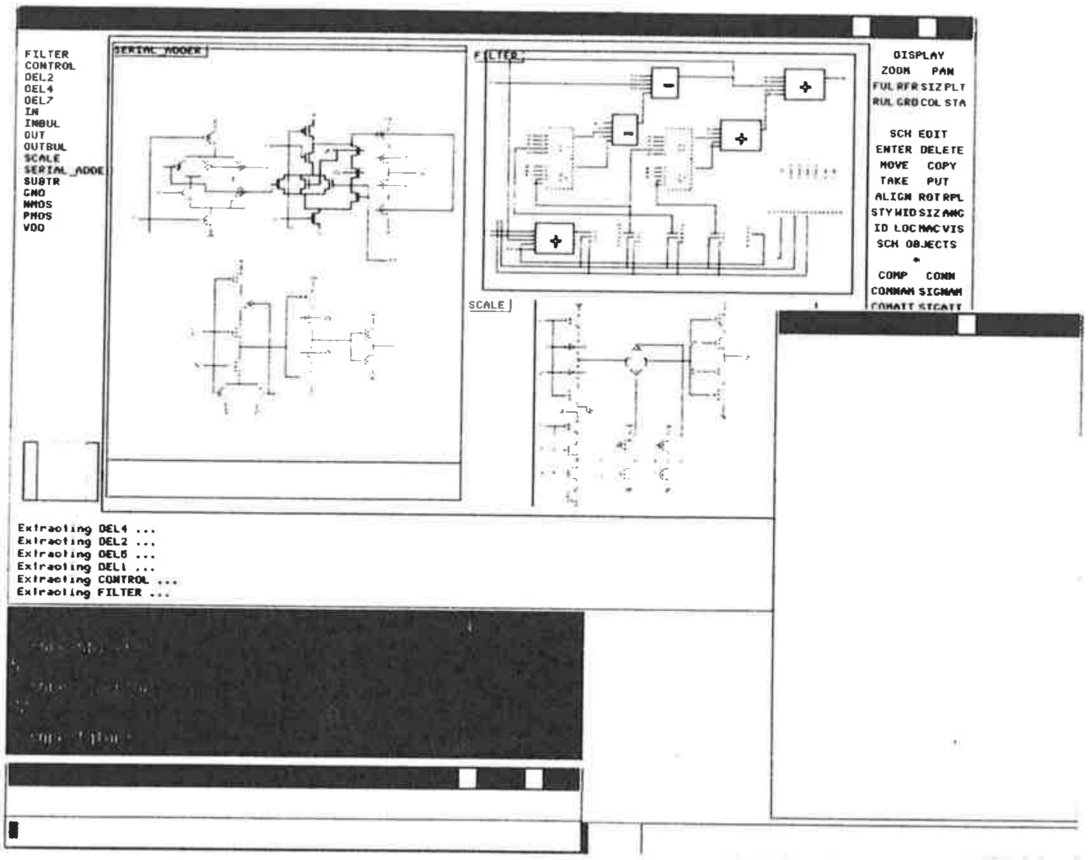
Mandatory interfaces . These interfaces are required if a bidirectional communication of the tool with an other tool is envisioned. It requires the interfacing to the following modules:

- **SPI** structure procedural interface to communicate (either receive or generate structure) with other tools.
- **DMS** Design Management System. To indicate which design objects are handled by the application program. This is required to know by the DTM what design objects are being handled by which programs.

Optional interfaces .

- **UIM** User interface module to have a uniform appearance to the user.
- **DBM** Database module to store the proprietary data of an application program on a design object.
- **Linker module** : this is only of importance for system design tools that require information of parameterized cells.

To allow highlighting of structural items in more cells, tools need to be able to be activated with more cells at once. Otherwise if highlights or pointing has to be done, the tools need to be started per cell. This would require too much processes.



```
Extracting DEL4 ...
Extracting DEL2 ...
Extracting DEL6 ...
Extracting DEL1 ...
Extracting CONTROL ...
Extracting FILTER ...
```

Figure 4: Interactive communication between schematics capture system CASS and the rule-based electrical verification system DIALOG

6 Current implementation examples.

The SPI interface has been used on VAX/VMS for the interactive interface between the SLOCOP [10] timing verifier and the CAMELEON [14,15] symbolic layout system. This allows to highlight critical delay paths directly in the generated layouts.

The current implementation of this system architecture is under development on Apollo workstations. To improve portability as much as possible use is made of standards such as UNIX, TCP-IP, UNIX sockets for interprocess communication.

Figure 4 illustrates the interactive coupling between two separate programs: schematics editing system CASS [22] and the rule-based electrical verification system DIALOG [7]. The interactive communication between the two tools allows that the designer can point to structural items in the schematics editor window and pass this information directly to the DIALOG verification program. DIALOG on its turn can give information to the designer by highlighting structural objects in the schematics editor. Notice that this is performed without actually stopping either of the programs.

The integration of DIALOG with the CASS schematics editor also illustrates the open character of the system architecture presented here. The two tools have their own database structures. They are written in different computer languages (LISP and FORTRAN). They operate in different processes. The main requirement for integration has been that the tools had to be interfaced to the SPI structure procedural interface.

7 Summary.

In this paper, a *flexible and open* framework for *interactive* CAD tools has been presented. The emphasis is put on the *interactivity* of the CAD verification tools. This is required to be able to communicate in an interactive way with the *graphical* design tools a designer is confronted with. In this way the design cycle is shortened and verification tools can have closer communication with the designer. An example of such a tool is DIALOG [7] that can directly, while doing its analysis, indicate possible design errors in the schematics. Another representative analysis tool is the SLOCOP [10] timing verifier that can directly indicate critical delay paths in the schematics or symbolic layout by immediate highlighting, without stopping either the timing verifier or the schematics- or the symbolic layout editor. As less as possible constraints have been put on the tools themselves to be able to integrate already existing tools and tools obtained from outside with minor efforts.

8 Appendix: The SPI Structure Procedural Interface.

The structure procedural interface SPI 1.6 is described in its specification[19]. Currently language bindings are defined for C and for Pascal. Additional bindings have been set up for FORTRAN and LISP. In this appendix the C-bindings are explained.

This appendix gives a short summary of the SPI procedures as they are available in SPI 1.6.

SPI procedures are provided for the following functions:

- control
- request structural information
- request extra information as attributes, and backannotate
- highlight structural objects
- request selection of a structural object by the user
- ask for user names of structural objects

In the following subsections the main procedures for the above functions are described briefly.

8.1 Control

SPI consists of procedures called from a structure receiver and implemented in a structure generator (can be in an other process via interprocess communication protocols). Therefor the structure generator is conceptually viewed as a submodule of the receiver. The following procedures allow the receiver to initialize this module, activate it, and clean up before halting the program.

`void` **SPIinit**`()`

parameters: None.

returns: Nothing.

usage: This function initializes the structure generator. It must be called exactly once before calling any other SPI procedure.

`void` **SPInewStart**`()`

parameters: None.

returns: Nothing.

usage: This function will activate the structure generator to do its job. It must be called before starting SPI communication on a new structure. It may be called any number of times in each program run. Note: the structure in the structure generator can change during this call; therefor all old object identifiers become invalid! Object identifiers are explained in the next section.

`void` **SPIexit**`()`

parameters: None.

returns: Nothing.

usage: This function will do any operations necessary in the structure generator before halting the program. It must be the last SPI procedure call and must be called exactly once.

8.2 Structural Information

8.2.1 Terminology

The terms used for structural objects have been taken from the EDIF standard. A *cell* is a building block representing a part of the circuit. A cell communicates with the outside world through its *ports*. A cell can either be a leaf cell or be composed of *instances of* other cells. The ports of these instances and of the cell itself can be connected. A set of connected ports is called a *net*. Connections of instance ports are called internal connections. Connections of ports of the cell itself are called external connections.

8.2.2 Identification of structural objects

For efficiency reasons, all structural objects are referenced using arbitrary non-zero integer numbers instead of names. Each cell must have a unique identifier. Port, net and instance numbers must be unique within a cell. Instance ports do not have their own identifier; they are referenced using the instance number and the port number.

8.2.3 Representation of busses

Like all other structural information, bus ports and nets are represented explicitly in SPI using extra parameters in the structure communication procedures. For uniformity, every port is a bus port and every net a bus net. The width of a bus can either be fixed when the port or net is first referenced, or it can be left unspecified until connections are made. Normal ports and nets have a fixed bus width of 1. The wires in a bus are always numbered from 1 to the bus width. These numbers need not be equal to the user name for each wire, which can be an arbitrary string.

8.2.4 Declaration in C

`cellNr =` **SPIgetCell**`( name )`

parameters: `char *name` a cell name

returns: The number used to identify the cell with the specified name, or zero if such a cell does not exist in the structure generator.

usage: It is normally used to obtain the identifier of the top cell of a circuit; the identifiers of subcells are returned by SPIgetInst.

`portNr =` **SPIgetPort**`( cellNr, prevPortNr, width )`

parameters: `int cellNr` a cell identifier

`int prevPortNr` the identifier of a port of that cell, or zero.

`int *width` the address of an integer to receive the bus width of the next port. The bus width is 1 for normal ports, and zero for ports with an unspecified bus width. A structure receiver that cannot process busses should set this parameter to NULL.

returns: The identifier of the next port. If "prevPortNr" is zero, the first port will be returned. If "prevPortNr" corresponds to the last port, zero will be returned.

usage: Normally used to read all the ports of a cell one by one. The "prevPortNr" argument makes sure that the function has no side effects.

`netNr =` **SPIgetNet**`( cellNr, prevNetNr, width )`

parameters: `int cellNr` a cell identifier

`int prevNetNr` the identifier of a net of that cell, or zero.

`int *width` the address of an integer to receive the bus width of the next net. The bus width is 1 for normal nets, and zero for nets with an unspecified bus width. A structure receiver that cannot process busses should set this parameter to NULL.

returns: The identifier of the next net. If "prevNetNr" is zero, the first net will be returned. If "prevNetNr" corresponds to the last net, zero will be returned.

usage: Normally used to read all the nets of a cell one by one. The "prevNetNr" argument makes sure that the function has no side effects.

`instNr =` **SPIgetInst**`( cellNr, prevInstNr, instOf )`

parameters: `int cellNr` a cell identifier

`int prevInstNr` the identifier of a inst of that cell, or zero

`int *instOf` the address of an integer to receive the identifier of the cell of which the next instance is an instance.

returns: The identifier of the next instance. If "prevInstNr" is zero, the first instance will be returned. If "prevInstNr" corresponds to the last instance, zero will be returned.

usage: Normally used to read all the instances of a cell one by one. The "prevInstNr" argument makes sure that the function has no side effects.

`netNr =` **SPIgetIntConn**`( cellNr, instNr, portNr, portIndex, netIndex, width )`

`netNr =` **SPIgetExtConn**`( cellNr, portNr, portIndex, netIndex, width )`

parameters: `int cellNr` a cell identifier

`int instNr` the identifier of an instance in that cell

`int portNr` the identifier of a port of that instance `int *portIndex` the address of an integer used to loop through the wires of a bus port. Set it to zero for the first call. The structure generator will increment it, and return the net and netIndex for the resulting portIndex. When there are no more wires in the bus port, the structure generator will reset the portIndex to zero. A structure receiver that cannot process busses can set this parameter to NULL.

`int *netIndex` the address of an integer to receive the index in the net to which the port is connected. A structure receiver that cannot process busses can set this parameter to NULL.

`int *width` the address of an integer to receive the width of the connection. This is the number of wires of the current port and net that are connected in sequence: current portIndex to current netIndex, next portIndex to next netIndex, and so on. This can be used as a shortcut for regular bus connections. Connections in reverse order (next portIndex to previous netIndex ...) can be specified with a negative value for width. If the structure receiver uses this shortcut, it must increment portIndex accordingly for the next call of SPIgetIntConn or SPIgetExtConn. A structure receiver that cannot handle this shortcut for busses can set this parameter to NULL.

returns: The identifier of the net to which the port is connected, or zero if the port is not connected.

usage: Used to read the internal or external connections of a cell. In the general case, bus connections can be complex; therefor a few examples are given below.

1. Simple non-bus connections: if the port specified by (instNr,portNr) is a normal port connected to a normal net, portIndex, netIndex and width will all return 1 and the function will as always return the net number.

2. Simple bus connection: if the port is a bus port N wide, and it is connected to a bus net also N wide, and portIndex is zero on input, the following values can be returned: portIndex=1, netIndex=1, width=N. This connects all the wires of the bus at once. Alternatively, each wire can be returned separately: the first call can return portIndex=1, netIndex=1, width=1, the second (with portIndex=1 on input) can return portIndex=2, netIndex=2, width=1 and so on. The last call, with portIndex=0 on input, will return portIndex=0. The second alternative is less efficient but more general: separate wires of the bus port can be connected the other nets.

3. Inverted bus connection: if the first wire of the bus port is connected to the N'th wire of the net, the second port wire to the N-1'th net wire, and so on, the following values can be returned: portIndex=1, netIndex=N, width=-N.

8.3 Attributes

Extra information associated with a cell, port, instance, instance port or net is called an attribute. Attributes have a name and a value. The name is represented as a character string. The value can be a character string or a real or integer number.

SPI provides procedures to obtain the names of all available attributes of an object, to request the value of an attribute with a given name, and to backannotate an attribute value.

8.3.1 Attribute Names

```
name = SPIcellAttName( cellNr, prevName )
name = SPIportAttName( cellNr, portNr, prevName )
```

name = **SPInetAttName**(cellNr, netNr, prevName)
name = **SPIinstAttName**(cellNr, instNr, prevName)
name = **SPIiprtAttName**(cellNr, instNr, portNr, prevName)

parameters: int cellNr, portNr, netNr, instNr identifiers for the object of which the attributes names are requested

char *prevName NULL or an existing attribute name

returns: the next attribute name. If prevName=NULL, returns the first attribute name. If prevName is the last attribute name, returns NULL.

usage: these functions are normally used to obtain all the attribute names associated with an object.

8.3.2 Attribute Values

value = **SPIcellAtt**(cellNr, name)
value = **SPIportAtt**(cellNr, portNr, name)
value = **SPInetAtt**(cellNr, netNr, name)
value = **SPIinstAtt**(cellNr, instNr, name)
value = **SPIiprtAtt**(cellNr, instNr, portNr, name)
value = **SPIcellIAtt**(cellNr, name)
value = **SPIportIAtt**(cellNr, portNr, name)
value = **SPInetIAtt**(cellNr, netNr, name)
value = **SPIinstIAtt**(cellNr, instNr, name)
value = **SPIiprtIAtt**(cellNr, instNr, portNr, name)
value = **SPIcellRAtt**(cellNr, name)
value = **SPIportRAtt**(cellNr, portNr, name)
value = **SPInetRAtt**(cellNr, netNr, name)
value = **SPIinstRAtt**(cellNr, instNr, name)
value = **SPIiprtRAtt**(cellNr, instNr, portNr, name)

parameters: int cellNr, portNr, netNr, instNr identifiers for the object of which the attribute value is requested

char *name the attribute name

returns: A pointer to a string, integer or real containing the attribute value. If an attribute with the specified name does not exist, NULL is returned.

usage: Read specific attribute values.

8.3.3 Backannotation

void **SPIcellPutAtt**(cellNr, name, value)
void **SPIportPutAtt**(cellNr, portNr, name, value)
void **SPInetPutAtt**(cellNr, netNr, name, value)
void **SPIinstPutAtt**(cellNr, instNr, name, value)
void **SPIiprtPutAtt**(cellNr, instNr, portNr, name, value)
void **SPIcellPutIAtt**(cellNr, name, value)

void **SPIportPutIAtt**(cellNr, portNr, name, value)
void **SPInetPutIAtt**(cellNr, netNr, name, value)
void **SPIinstPutIAtt**(cellNr, instNr, name, value)
void **SPIiprtPutIAtt**(cellNr, instNr, portNr, name, value)
void **SPIcellPutRAtt**(cellNr, name, value)
void **SPIportPutRAtt**(cellNr, portNr, name, value)
void **SPInetPutRAtt**(cellNr, netNr, name, value)
void **SPIinstPutRAtt**(cellNr, instNr, name, value)
void **SPIiprtPutRAtt**(cellNr, instNr, portNr, name, value)

parameters: int cellNr, portNr, netNr, instNr identifiers for the object of which the attribute value is backannotated

char *name the attribute name

char *value the attribute string value

int value the attribute integer value

float value the attribute real value

returns: Nothing.

usage: Backannotate.

8.4 Highlight Operations

void **SPIhiliteCell**(cellNr, color, reason)
void **SPIhilitePort**(cellNr, portNr, color, reason)
void **SPIhiliteNet**(cellNr, netNr, color, reason)
void **SPIhiliteInst**(cellNr, instNr, color, reason)
void **SPIhiliteIprt**(cellNr, instNr, portNr, color, reason)

parameters: int cellNr, portNr, netNr, instNr identifiers for the object to be highlighted

int color specifies the highlight color to be used. This parameter can have the following values:
SPI_UNHILITE Unhighlight, redraw object in default color
SPI_HILITE_1 Highlight in color number 1
SPI_HILITE_2 Highlight in color number 2
SPI_HILITE_3 Highlight in color number 3
SPI_HILITE_4 Highlight in color number 4
SPI_INVISIBLE Make invisible, redraw object in background color
SPI_BLACK
SPI_WHITE
SPI_RED
SPI_GREEN
SPI_BLUE
SPI_CYAN
SPI_MAGENTA
SPI_YELLOW
SPI_RED_YELLOW

SPI_GREEN_YELLOW
SPI_GREEN_CYAN
SPI_BLUE_CYAN
SPI_BLUE_MAGENTA
SPI_RED_MAGENTA
SPI_DARK_GRAY
SPI_LIGHT_GRAY
For colors number 1 to 4, the structure generator may choose any color (or line type or other highlight method) different from the default and background colors. The explicit colors are only provided for those tools that want explicit control over the highlight color. The structure generator may or may not implement them, and nothing guarantees that any of these colors is different from the default or background color.

`char *reason` a string explaining the reason for the highlight operation; this string should be displayed to the user by the structure generator.

returns: Nothing.

usage: Highlight and unhighlight objects.

8.5 Select Operations

`cellNr =` **SPIselectCell**`( reason )`
`portNr =` **SPIselectPort**`( cellNr, reason )`
`netNr =` **SPIselectNet**`( cellNr, reason )`
`instNr =` **SPIselectInst**`( cellNr, reason )`
`iprtNr =` **SPIselectIprt**`( cellNr, instNr, reason )`

parameters: `int cellNr` identifier of the cell in which an object must be selected

`char *reason` a string explaining the reason for the select operation; this string should be displayed to the user by the structure generator.

returns: The identifier of the selected object. If the user explicitly does not select an object, zero will be returned. This feature can be used to terminate the selection of a list of objects.

usage: Allow user selection of objects.

8.6 User names of structural objects

`name =` **SPIcellName**`( cellNr )`
`name =` **SPIportName**`( cellNr, portNr, index )`
`name =` **SPInetName**`( cellNr, netNr, index )`
`name =` **SPIinstName**`( cellNr, instNr )`

parameters: `int cellNr, portNr, netNr, instNr` identifiers for the object of which the name is requested `int` index the bus index. If this parameter is zero, the full bus name will be returned (eg "DATA$< 4 : 1 >$"). Otherwise, the name of a wire in the bus will be returned (eg "DATA$< 2 >$" for a bus "DATA$< 4 : 1 >$" with index=3).

returns: The user name of the object

usage: Normally, the communication between a program coupled to SPI and its user will use highlight and select operations. Sometimes however, it is useful to know the user name of some or all structural objects. This is where the above functions are useful.

For hierarchical circuits, these procedures return simple object names. For example, "OUT" will be returned for port OUT of cell NAND. For expanded circuits, these procedures return the full path name, excluding the name of the top cell. For example, "ALU/NAND1/OUT" will be returned for port OUT of instance NAND1 in instance ALU in cell CPU. These full path names will be constructed by the hierarchy expander.

9 SPI Utilities

9.1 The Hierarchy Expander

SPI can be used for hierarchical and for expanded circuits. When a hierarchical editor has to be interfaced to a tool that processes only flat circuits, an SPI hierarchy expander is needed.

The SPI hierarchy expander HEX uses SPI for both its hierarchical input and expanded output. It does not only expand the structure, but also translates highlight and select operations on the expanded circuit to hierarchical highlight and select operations.

With HEX, the structure generator (editor) implements the SPI procedures just as before. The structure receiver (simulation or verification tool) calls the SPI procedures, but with a different prefix (HEX instead of SPI).

9.2 The Bus Expander

SPI can be used for circuits with or without busses. When an editor supporting busses has to be interfaced to a tool not supporting busses, an SPI bus expander is needed.

The SPI bus expander BEX uses SPI for both its input and output. With BEX, the structure generator (editor) implements the SPI procedures just as before. The structure receiver (simulation or verification tool) calls the SPI procedures, but with a different prefix (BEX instead of SPI).

9.3 SPI-to-file and file-to-SPI modules

In some situations, direct communication between a structure generator and receiver is not possible. This happens for example when the generator and receiver operate on a different platform and no communication network is available between them. In these cases, the SPI-to-file and file-to-SPI modules can be used.

The SPI-to-file module is a structure receiver that creates a text file containing a description of a circuit including all its attributes. The file-to-SPI module is a structure generator that reads such a text file. It does however not implement highlight- and select operations.

References

[1] K.H.Keller, "An Electronic Circuit CAD Framework", *Memorandum No. UCB/ERL M84/54, Ph.D Dissertation, University of California, Berkeley*, 6 July 1984.

[2] D.S. Harrison, P.Moore, R.L.Spickelmier, A.R.Newton, "Data Management and Graphics Editing in the Berkeley Design Environment", *IEEE International Conference on Computer-Aided Design ICCAD-86*, November 11-13, 1986 Santa Clara CA., pp.24-27.

[3] Jaan Haabma, "NMP-CAD Base System", *Proceedings IFIP WG 10.2 workshop: "Tool Integration and Design Environments"*, edited by F.J.Rammig, North-Holland, 1988.

[4] K.Gootheil, G.Kachel, t.Kathoefer, H.J.Kaufmann, B.Kleinjohann, E.Kupitz, J.Miller, B.Nelke, F.J.Rammig, B.Steinmueller, C.White, "The Cadlab Workstation CWS - An Open System for Tool Integration", *Proceedings IFIP WG 10.2 workshop: "Tool Integration and Design Environments"*, edited by F.J.Rammig, North-Holland, 1988.

[5] L.-P.Demers, P.Jacques, S.Fauvel, E.Cerny, "An Approach to Object-Oriented Integration of VLSI Tools", *Proceedings IFIP WG 10.2 workshop: "Tool Integration and Design Environments"*, edited by F.J.Rammig, North-Holland, 1988.

[6] L.Claesen, H.De Man, I.Bolsens, W.De Rammelaere, D.Dumlugol, P.Lammens, P.Odent, R.Severyns, E.Vanden Meersch, "Electrical, Timing and Behavioral Verification in the Meet-in-the-Middle MOSVLSI Design Environment of CATHEDRAL-II", *Proceedings IEEE International Conference on Computer Design: VLSI in Computers & Processors, ICCD-87*, Port Chester, New York, Oct.5-Oct-8, 1987.

[7] H.De Man, I.Bolsens, E.Vanden Meersch, J.Van Cleynenbreughel, "DIALOG: An Expert debugging System for MOSVLSI Design", *IEEE Transactions on Computer Aided Design*, CAD-4, No.3, June 1985, pp. 303-311.

[8] J.K.Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-4, no.3, pp. 336-348 July 1985.

[9] N.P.Jouppi, "Timing Analysis and Performance Improvement of MOS VLSI Designs", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No.4, pp. 650-665, July 1987.

[10] E.Vanden Meersch, L.Claesen, H.De Man, "SLOCOP: A Timing Verification Tool for Synchronous CMOS Logic", *Proceedings European Solid State Circuits Conference*, ESSCIRC'86, Delft, September 16-18, 1986.

[11] J.Benkoski, E.Vanden Meersch, L.Claesen, H.De Man, "Efficient Algorithms for Solving the False Path Problem in Timing Verification", *Digest of technical papers IEEE International Conference on Computer-Aided Design ICCAD-87*, Santa Clara CA, November 9-12, 1987.

[12] D.Dumlugol, P.Odent, J.Cockx, H.De Man, "The Segmented Waveform Relaxation Method for Mixed-Mode Switch-Electrical Simulation of digital MOS VLSI Circuits and its Hardware Acceleration on Parallel Computers", *Proceedings IEEE Conf. ICCAD'86*, Santa Clara, CA, Nov. 1986, pp. 84-87.

[13] L.M.Vidigal, S.R.Nassif, S.W.Director, "CINNAMON: Coupled Integration and Nodal Analysis of MOs Networks", *23rd Design Automation Conference*, June 29-July 2, 1986, pp.179-185.

[14] K.Croes, L.Rijnders, "CAMELEON: A Technology Independent Symbolic Layout System", *Internal Report IMEC, MR03KUL-7-B3-2*, January 1986.

[15] K.Croes et al.,: "CAMELEON, a process tolerant symbolic layout system", *Digest of technical papers ESSIRC-87*, Bad Soden, Germany.

[16] I.Vandeweerd, "Module Generation Environment: reference manual", *Internal Report ESPRIT1058/IMEC/6.86/D8619.*

[17] P.Six, L.Claesen, J.Rabaey, H.De Man, "An Intelligent Module Generator Environment", *Proceedings of 23rd Design Automation Conference*, Las Vegas, June 29-July 2, 1986, pp. 730-735.

[18] L.Claesen, Ph.Reynaert, G.Schrooten, "Open system architecture for communicating CAD tools", *Report ESPRIT1058/IMEC/12.86/D8652*, IMEC Leuven, Belgium.

[19] J.Cockx, Ph.Reynaert, "ESPRIT-1058: SPI Specification", *Report ESPRIT1058/SL/12.86/D8654*, IMEC Leuven, Belgium.

[20] R.L.Spickelmier, "Verification of Circuit Interconnectivity", *Report Electronics Research Laboratory*, Univ. of California Berkeley, June 1983.

[21] E.Vanden Meersch, R.Severyns, "HILARICS: User's Manual, 2nd edition", *Internal report IMEC, MR03-KUL-7-B3-2*, January 1986.

[22] –, "CASS The Computer-Aided Schematic System : User's Guide", *Document no. M-002-3*, Silvar-Lisco B-3030 Leuven Belgium.

[23] N. van der Meijs, T.G.R. van Leuken, P. van der Wolf, I.Widya, P. Dewilde, "A Data Management Interface to Facilitate CAD/IC Software Exchanges", *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers*, ICCD'87, New York, Oct.5-8, pp. 403-406.

[24] P. van der Wolf, "Data Management for VLSI Design: Conceptual Modeling, Tool Integration & User Interface", *Proceedings IFIP WG 10.2 workshop: "Tool Integration and Design Environments"*, edited by F.J.Rammig, North-Holland, 1988.

[25] A. Di Janni, "A Monitor for Complex CAD Systems", *Proc. 23rd ACM/IEEE Design Automation Conference*, June 29 - July 2, 1986, pp. 145-151.

IFIP WG 10.2 Workshop on
Tool Integration and Design Environments
Paderborn, FRG, 26-27 November, 1987

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

TOOL INTEGRATION AND DESIGN ENVIRONMENTS

Proceedings of the IFIP WG 10.2 Workshop on
Tool Integration and Design Environments
Paderborn, FRG, 26-27 November, 1987

edited by

Franz J. RAMMIG
University of Paderborn
Paderborn, FRG

1988

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO