SCHEM: PARAMETERIZABLE SCHEMATIC ENTRY SYSTEM USING AUTOMATIC SYMBOL GENERATION

by

Jonathan S. Min

Memorandum No. UCB/ERL M89/73

23 May 1989

SCHEM: PARAMETERIZABLE SCHEMATIC ENTRY SYSTEM USING AUTOMATIC SYMBOL GENERATION

by

Jonathan S. Min

Memorandum No. UCB/ERL M89/73

23 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

SCHEM: PARAMETERIZABLE SCHEMATIC ENTRY SYSTEM USING AUTOMATIC SYMBOL GENERATION

by

Jonathan S. Min

Memorandum No. UCB/ERL M89/73

23 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

SCHEM: Parameterizable Schematic Entry System Using Automatic Symbol Generation

Jonathan S. Min

Department of Electrical Engineering
and Computer Science
University of California, Berkeley

ABSTRACT

A parameterizable OCT/VEM/RPC-based schematic capture system, called SCHEM, is introduced in this report. SCHEM is developed as a graphical interface for LagerIV, a silicon compiler system. LagerIV uses OCT as a central database and generates a final layout from an architectural description input through several phases of design synthesis. Up to now, only a textual description of the architecture has been available as the input to the LagerIV system. A better user interface, using VEM schematic entry, is implemented in order to directly generate the OCT database.

May 23, 1989

Table of Contents

SCHEM: Parameterizable Schematic Entry System Using Automatic Symbol Generation

Jonathan S. Min

Department of Electrical Engineering
and Computer Science
University of California, Berkeley

1. Introduction

The goal of this project is to develop a schematic entry system for the LagerIV silicon compiler. To use the LagerIV system[Rab85a], a designer had to write structural description language(SDL) files for leafcells and macro cells. The syntax of the SDL file is lisp-like and includes four sections: parent-cell, layout-generator, sub-cells, and net-list. An example of a SDL file describing the control circuit for a simple microprocessor is given in Fig. 1. This textual input approach, however, is not only tedious and error-prone but also gives little insight about the global topology of the circuit described. This might not be a problem for the experienced user, but, for the novice user, this serves as one of the biggest drawbacks of the LagerIV system. The obvious idea behind a graphical interface is the added commodity of visual assimilation of the design, along with the ease of interactive editing and language independence.

The hierarchical description of design and parameterization of both module parameters and busses are key issues when implementing this schematic tool. SCHEM uses a schematic symbol generator to automatically create a black box symbol for the next level of design hierarchy. The notion of terminal expander, compared to bus splitter, is introduced to handle parameterized busses. Also, the maintenance and interface of the cell library, as well as user-friendliness, are greatly emphasized.

2. OCT, VEM and RPC

SCHEM uses the data structure of OCT, the graphical capabilities of VEM and the modular code independency offered by RPC compilation. OCT is the Berkeley VLSI/CAD data manager.[Moo87a]. A design has three parts associated with it: namely, *cell:view:facet*. A cell

is a basic unit of a chip. It can be as small as a NAND gate, or as large as a CPU. A view is an aspect of a cell, depending on what design style one uses. The three main standard views are physical, symbolic, and schematic. The physical view is intended for a mask-level implementation of geometry, the symbolic view is very similar. It places instances of physical geometry without worrying about design rule constraints. The schematic view is intended for high level abstraction of design. This is suitable for the SCHEM implementation. Each view has a facet named "contents", which contains the actual definition of the view, and then may have an "interface" facet, which is the abstraction of the contents for hierarchy.

Some more OCT terminology is explained here to simplify further reading: *property, formal terminal, and bag*. A "property" is an attribute that can be attached to any object. The terminal in OCT is the connector for interconnection. If it is a terminal of the current facet, it is called the "formal terminal", and other facets may only reference it. The "bag" describes an object which is only used to hold other objects in OCT[Bur88a].

VEM(View Editor Monolith) is an X window-based interactive graphics environment for viewing and editing the OCT database. VEM has three different modes of viewing and editing as well: physical, symbolic, and schematic[Har86a]. SCHEM uses the VEM schematic editing window as its workspace. VEM also facilitates the use of various user application tools on OCT views through the RPC(Remote Procedure Call) package. In this way OCT and VEM work in the distributed environment[Spi87a]. Basically, SCHEM is an RPC application program that interacts with VEM to do graphical interface for LagerIV. The overview of the Berkeley CAD environment is shown in Fig. 2.

VEM is an on-going research system which is still evolving. Although VEM provides useful editing capabilities at present, it still lacks some important editing utilities. Some of them will be available in the next version of VEM -- VEM7:

a) Mouse-driven Instantiation
b) Manhattan Instance Drag
c) Undo capability
d) Displayable Property(Label)

Even though commercial graphical editors such as ViewLogic provide better editing capabilities, they have some shortcomings, which are inherent to the way they handle their data presentation.

```
; sdl-file for the description of the controller using logic optimization
;
(parent-cell controller)
(parameters inputs outputs bdsyn_file)
(layout-generator Flint)
;
(subcells (fsm_bdsyn FSM
                ((inwidth inputs) (outwidth outputs) (bdsyn bdsyn_file))
          )
)
;
; internal interconnection
;
(net State
  ( (FSM OUT 4) (FSM IN 3) )
)
;
; formal (boundary) terminal definition and interconnection
;

(net INST0 ( (parent INST0) (FSM IN 0) ))
(net INST1 ( (parent INST1) (FSM IN 1) ))
(net CARRY ( (parent CARRY) (FSM IN 2) ))
(net LOAD  ( (parent LOAD)  (FSM OUT 0) ))
(net SEL1  ( (parent SEL1)  (FSM OUT 1) ))
(net SEL2  ( (parent SEL2)  (FSM OUT 2) ))
(net DATAV ( (parent DATAV) (FSM OUT 3) ))

;
;NOTE: we have to also create clock, ground and supply terminals
; and connect them to correspoding terminals
;
; Clocks
;
(net PHI1 (NETTYPE CLOCK) ( (parent PHI1) (FSM PHI1) ))
(net PHI2 (NETTYPE CLOCK) ( (parent PHI2) (FSM PHI2) ))
;
; Supply nets
;
(net GND (NETTYPE GROUND) ( (parent GND) (FSM GND) ))
(net Vdd (NETTYPE SUPPLY) ( (parent Vdd) (FSM Vdd) ))

(end-sdl)
```

Fig. 1 a SDL file example

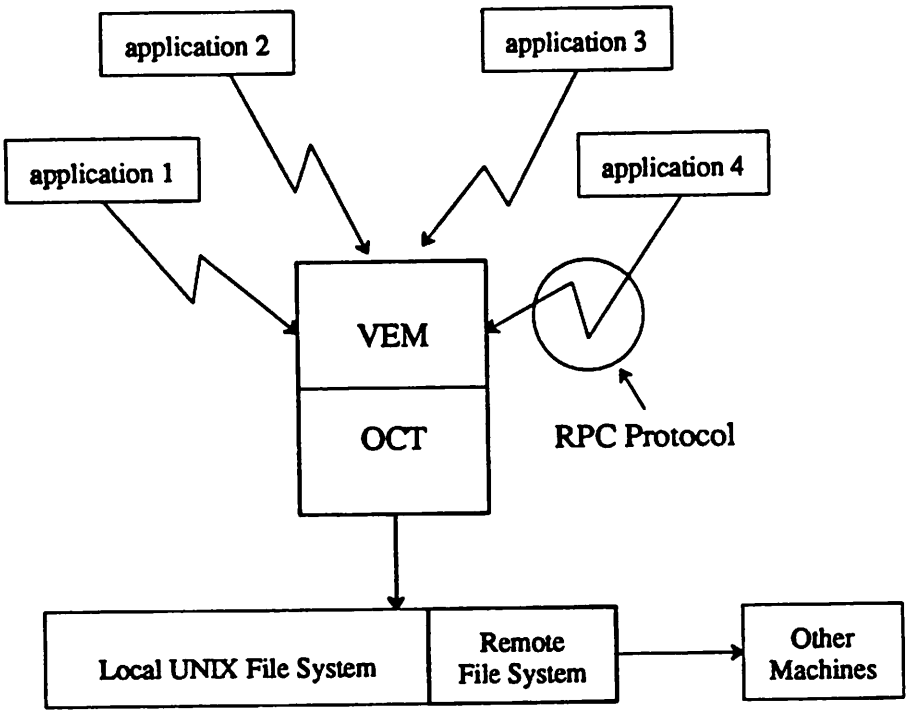

Fig. 2 the Berkeley CAD environment

Also, since the menus are fixed, the programmer can not customize them. The introduction of parametrization and hierarchical design is available within the OCT/VEM/RPC environment, in which the programmer has direct access to the central database and can expand the menus according to his need. VEM also supports multiple windows per design, which is a necessity for a large size design. In short, provided that a better user-interface all soon be available, VEM has proven to be the solution for the LagerIV graphical interface. Since the LagerIV system uses OCT as its central database and the user needs VEM to view physical layout designs anyway, the OCT/VEM/RPC approach has provided unified integration for the whole LagerIV system.

3. Overview of LagerIV

LagerIV is an integrated automated chip design system that consists of a set of layout generation tools and a set of MOSIS SCMOS cell libraries. It currently provides three different layout generation tools: stdcell(MSU stdcell layout placer/router), TimLager(Tiler for module layout

generator), and Flint(macrocell placer/router), and also has library cells support for Stdcell, Tim-Lager, and dpp(a structure preprocessor for bitslice datapaths)[Jai88a]. The Lager system provides many desirable features important in ASIC(Application Specific Integrated Circuit) design, such as parameterizability and modularity. The parameterizability refers to an object-oriented style which uses one generic cell for many other applications, changing only its parameters[Rab85a].

An overview of the OCT database framework assumed by LagerIV is given in Fig. 3.1.

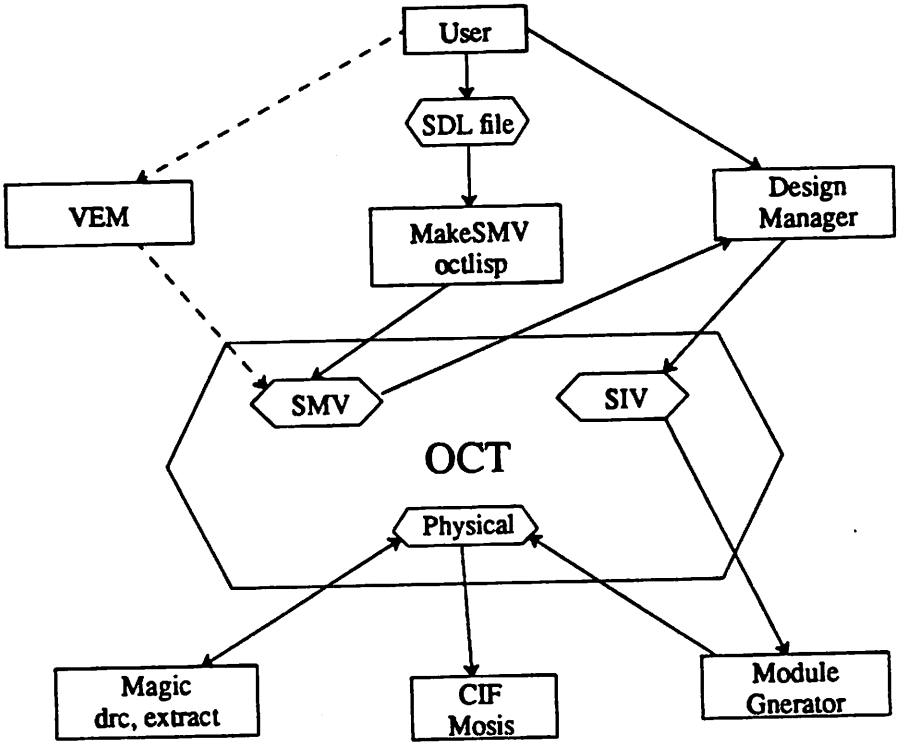

Fig. 3.1 LagerIV Operation Procedure

There are two structural OCT views associated with LagerIV, beside the physical layout view: the Structure_master view(SMV) & Structure_instance view(SIV)[Jai88a]. These views do not have any geometry information, but they keep all the other information pertinent to the design, such as connectivity, vectorization, and hierarchy. The SMV represents the parameterized architecture of the chip to be designed, and its essential features are shown in Fig. 3.2. It keeps its module parameters in the "FORMAL_PARAMETERS" bag, and net/terminal vectorization information

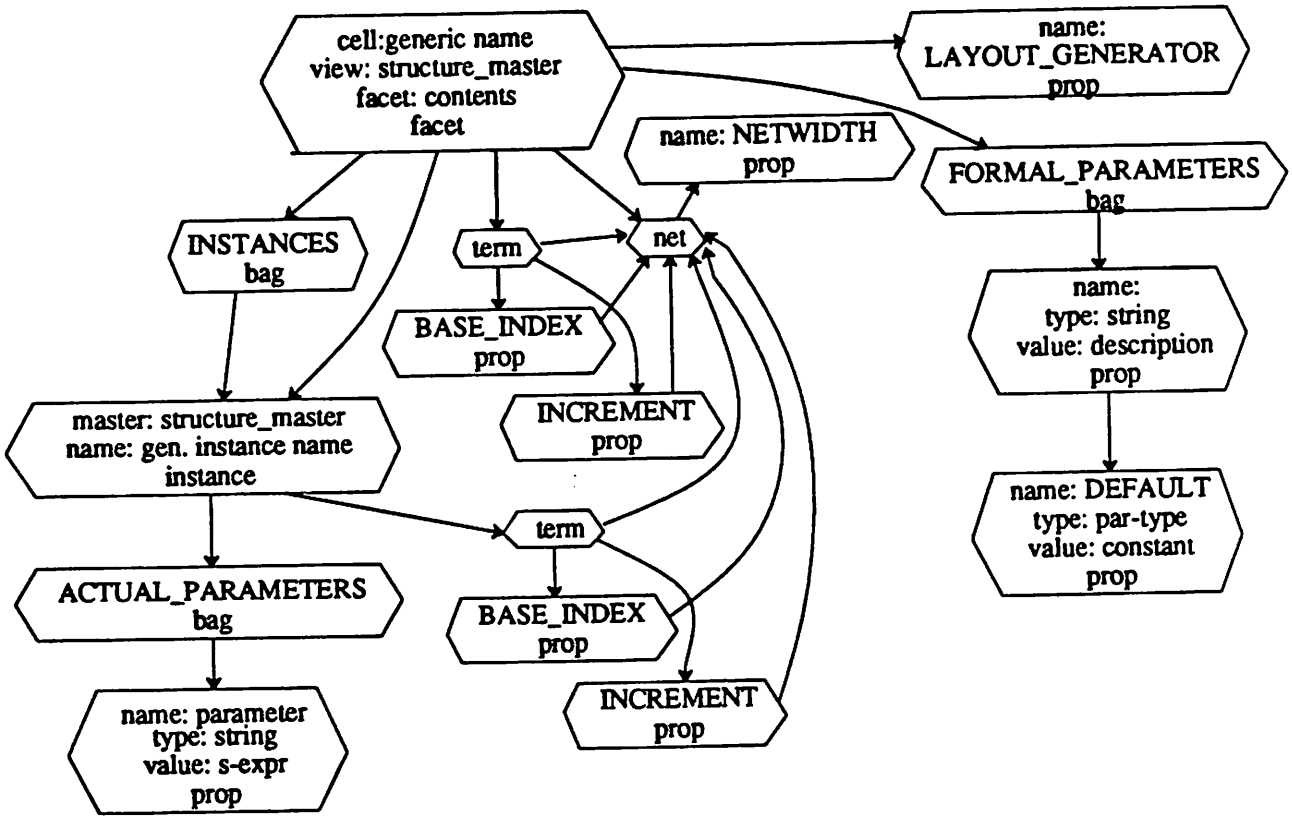

Fig. 3.2 the Structure_master View

in properties called BASE_INDEX, INCREMENT, and NETWIDTH. These properties are necessary to provide a parameterizable bus structure. The SIV shown in Fig. 3.3 represents the architecture of a given instance of a chip.

DMoct, the design manger of LagerIV, is the user interface to LagerIV. Given the parameter values specified by the user, it creates the SIV from the SMV and automatically executes various layout tools to generate a physical chip layout. Essentially DMoct accepts parameter values for the top level and passes them to its subcells using the parameter relations in the "ACTUAL_PARAMETERS" bag. In addition, the nets and terminals that have NETWIDTH, BASE_INDEX and INCREMENT properties attached in the SMV are expanded according to their final values. It is assumed to provide the input information to all module layout generators. The hierarchical description of circuit design is flattened in the SIV creation phase by DMoct, given the SMVs of subcells.

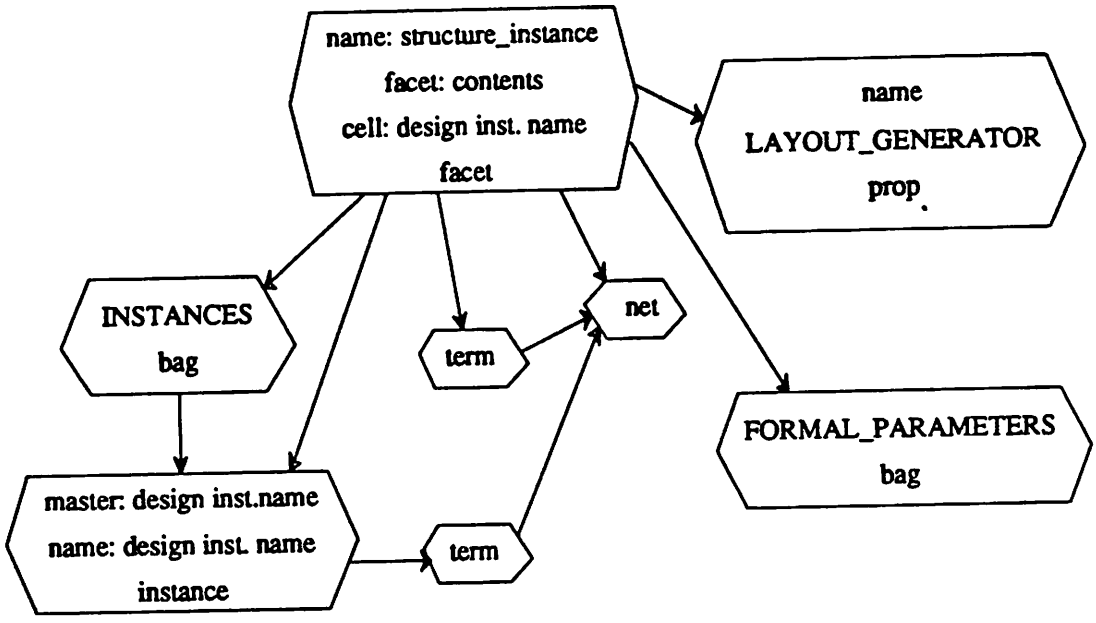

Fig. 3.3 the Structure_instance View

4. Implementation of SCHEM

4.1. Overview (Key Issues)

The task here is to make a schematic interface from VEM to the OCT SMV directly. The VEM schematic editing provides a limited utility to create vectorization between net/terminal and other parameters. Remote-application routines are written to make this graphical interface more user-friendly. Major features of SCHEM are explained in more detail in the following sections.

The realization of the project has been organized into three main modules. The first is the creation of the LagerIV library cells, building blocks, from which the user is allowed to choose when designing a circuit. The second part of the project is the development of new user interface in VEM, specifically adapted to the information the user will be asked to enter while designing a circuit. The interface itself is built using the RPC package, offering portability as well as ease of modification. The final aspect of the project is an OCT Schematic view to SMV converter. These three functional partitions in SCHEM are shown in Fig. 4.1.

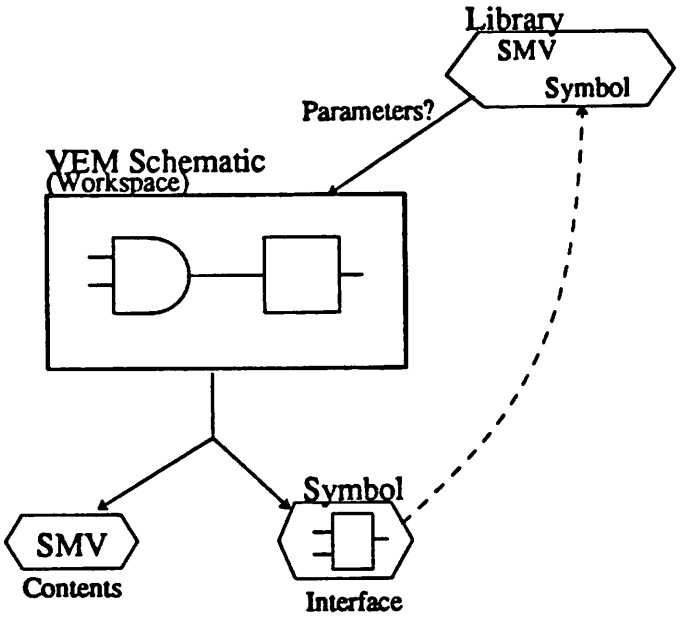

Fig. 4.1 Partitioning in SCHEM

Each design cell contains three views associated with it: schematic, SMV, and symbol. The schematic view is automatically created in VEM -- the workspace for the user when entering cell interconnections. The SMV has all the logical information pertaining to the LagerIV policy, but no geometry. The symbol contains only graphical information -- black box model with formal terminals for the next level of hierarchy. An example of the three views associated with the design is shown in Fig. 4.2: (a) the symbol view, (b) the SMV, and (c) the schematic view of the cell. The library consists of both leafcells and blocks which have the SMV and symbol view only. The schematic view is not needed for the library cell, since it is our starting block.

As the user gets a cell from the library, only the symbol is instantiated into the workspace in VEM. However, when the VEM schematic is processed to create the SMV of the design, this instance is referred to the SMV of the master cell, since this contains the actual contents. For example, the module parameters associated with the instance are searched from its SMV and prompted for its new values on instantiation. In other words, the SMV is the logical contents and the symbol is the interface. This partition is different from that of conventional OCT, which is done at the facet level. This division, done at the view level in SCHEM, has proven to be useful

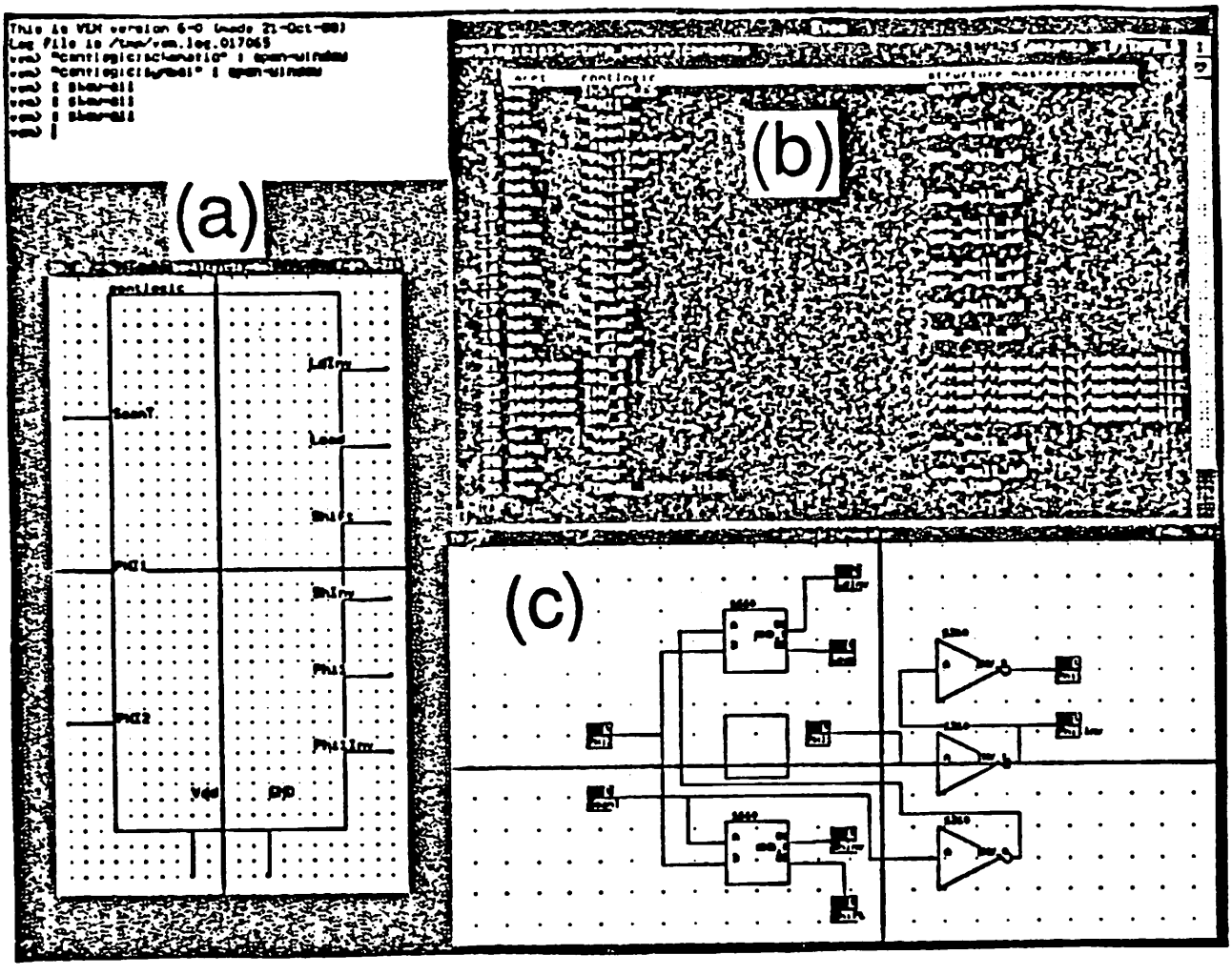

Fig. 4.2 an Example of the SMV, Symbol, Schematic view

because it avoids the tight binding between the contents and the interface that VEM enforces. As long as these two views have the same formal terminal names, whatever interconnection changes one make in the contents, the interface stays the same. Each cell can also have multiple symbols, if necessary.

4.2. Hierarchical Design using Automatic Symbol Generator

For each level of design, the black box symbol(interface) as well as the SMV(contents) is generated from the VEM schematic through SCHEM commands. This generated symbol with the unique label is used to represent the cell at the next level of design hierarchy. Separating the contents and interface in the way described above, the hierarchical description of the chip design of

LagerIV is fully supported. Refer to Fig. 4.3 for an example of a hierarchical design.

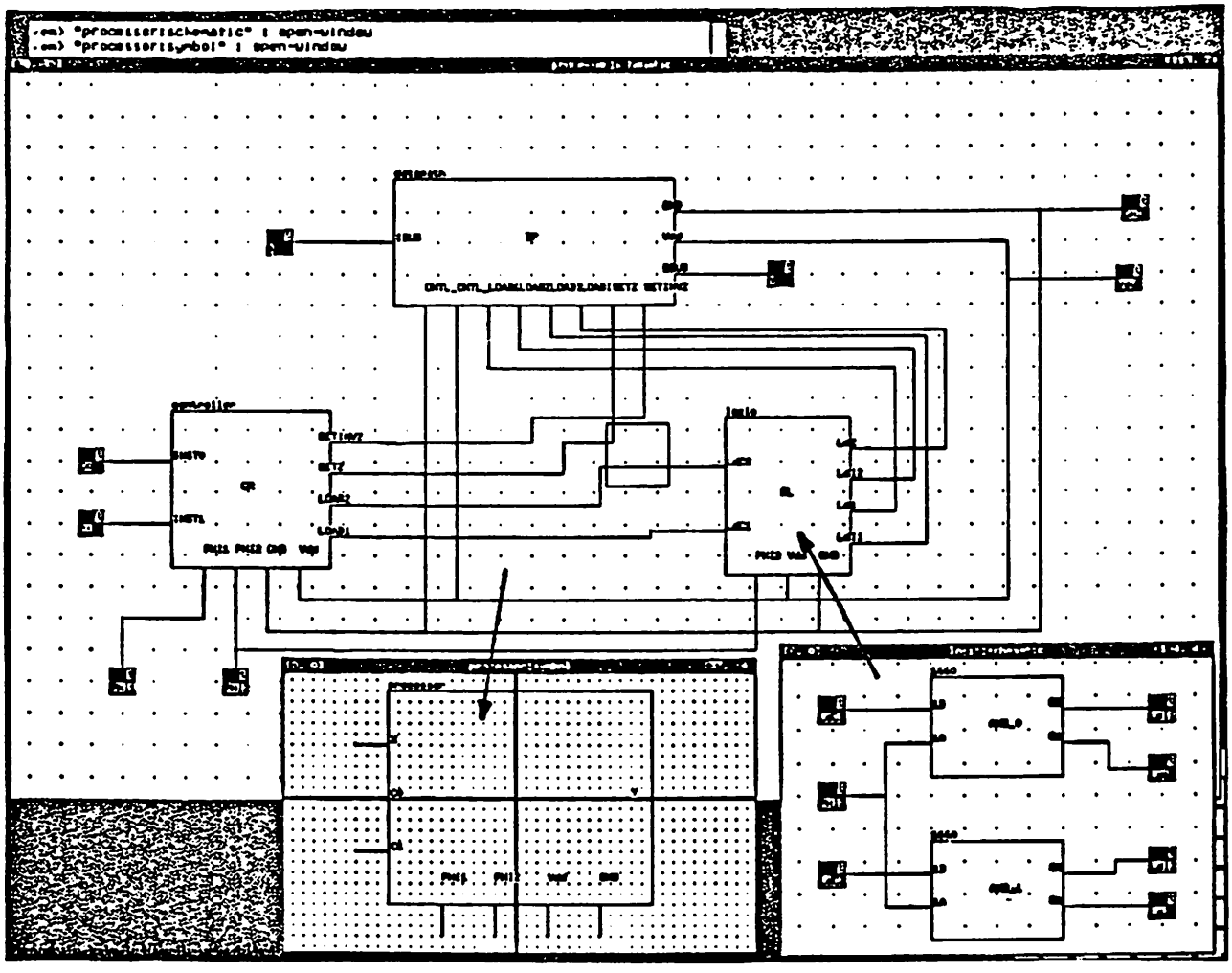

Fig. 4.3 an Example of the Hierarchical Design

The automatic symbol generator looks for every formal terminal in the VEM schematic, and prompts for the location of terminals -- left, right, top, or bottom. The size of the bounding box for the symbol is determined by the number of terminals on each side. This forces all schematic symbols to be of similar sizes, independent of the size of the contents. The size of the symbol for a 16-bit ALU and a 16-bit microprocessor should roughly be the same size even though the number of components they contain are quite different. Furthermore, there is a way to generate automatically the symbol for all the cells that already have SMVs. This turns out to be useful in creating the library symbols directly from their SMVs without going through the schematic view.

4.3. Parameterization

4.3.1. Module Parameters

The module parameters are stored in the "FORMAL_PARAMETERS" bag attached to the SMV of the subcell. When the subcell is instantiated, the user is prompted for the values of the parameters in a dialog box -- a customized X window available through the RPC package. The obtained values are then stored in the "ACTUAL_PARAMETERS" bag attached to the instance. The values can be either constants or formal parameters defined in the cell in current design. A typical example of a module parameter is *Num_Of_Bits* for the datapath cell.

4.3.2. Parameterized Bus

In SCHEM, the bus is also parameterized. In the macrocell layout, for instance, the user might need to specify the following condition:

> *The odd bits of bus X of block1 are connected to the even bits of bus Y of block2 and the even bits of bus X of block1 are connected to odd bits of bus Z of block3.*

The representation of parameterized busses in SDL format is shown in the SDL example of Fig. 1. For a net with NETWIDTH = N, for instance, values of BASE_INDEX = 0 and INCREMENT = 1 represent an even bit bus.

Graphically, a simple bus splitter as used in other schematic entry tools will not work. Here the net is logical, meaning that it does not have a final fixed value yet. Two sets of terminal and net pairs are actually needed to represent even and odd bits. The concept of the terminal expander is introduced to specify explicitly the presence of two parameterized nets. Refer to Fig. 4.4 for its implementation. Each net has its NETWIDTH, and every termExpander node has BASE_INDEX and INCREMENT properties associated with it, which are prompted for their values in a customized dialog box when the command is executed.

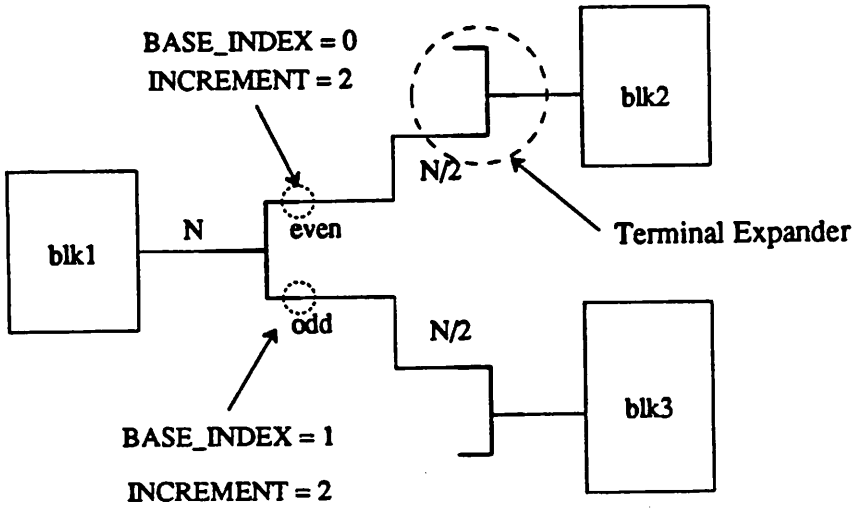

Fig. 4.4 the Terminal Expander

4.4. Library Management/Interface

As mentioned above, the LagerIV system has 3 cell libraries[Jai88a]:

a) TimLager library contains larger self-contained modules such as RAM, ROM, PLA, registers, and scan-latches.

b) Dpp library contains leafcells for building bit-sliced datapaths such as adders, multiplexers, shifters, and registers.

c) Stdcell library contains the MSU(Mississippi State University) standard cells for functions such as NAND and INVERTER.

All the schematic symbols for TimLager and dpp library cells are implemented as black box models using the automatic symbol generator program. Special attention is given to cells in stdcell library. Since most of these MSU library cells have a gate representation, a batch-oriented program is written separately to generate more meaningful symbols(refer to Fig. 4.5). The cell and terminal names are also labeled in the symbol. All the cell libraries are grouped and are available as three palettes in SCHEM, one for each library.

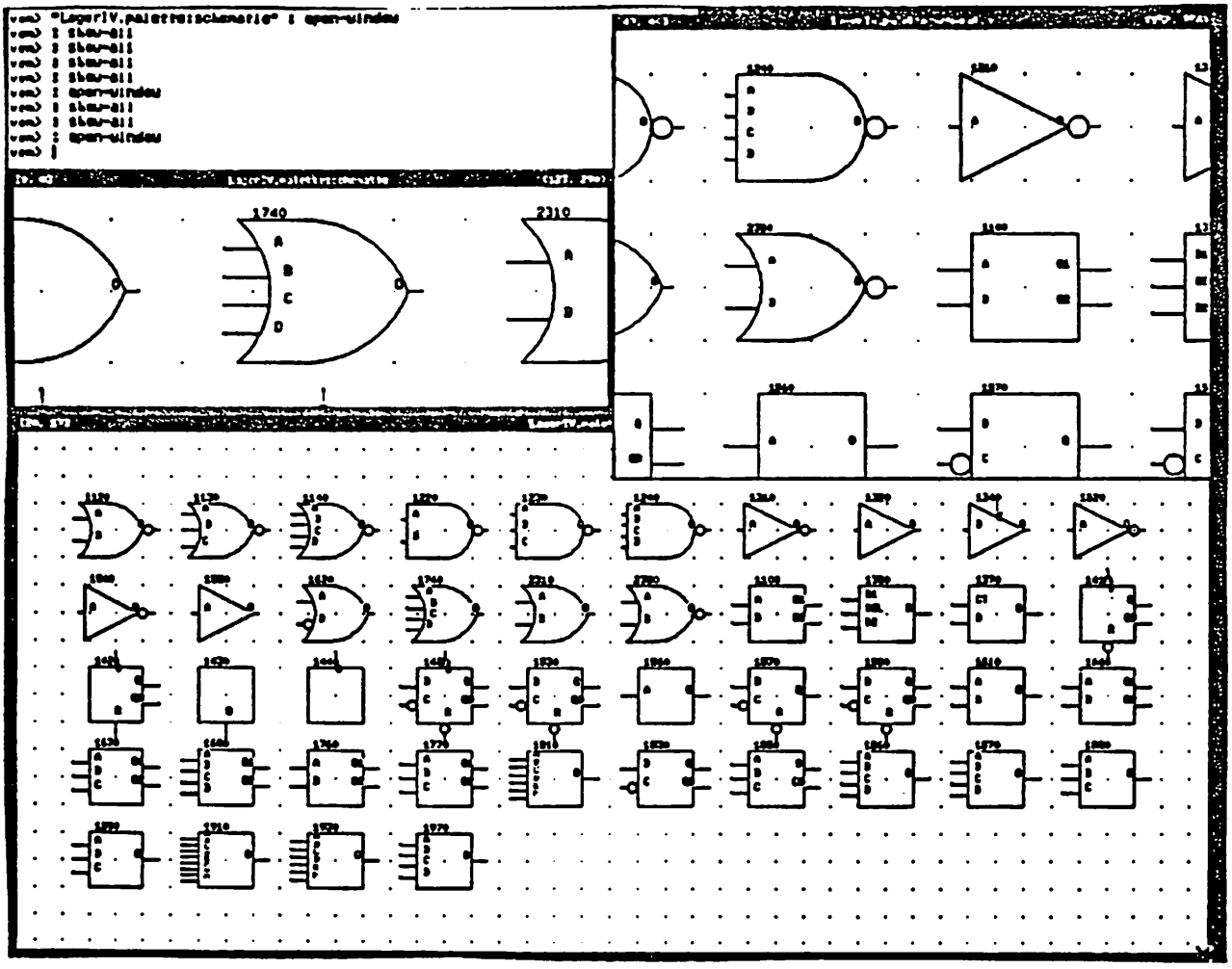

Fig. 4.5 Stdcell Library

The whole LagerIV cell library is organized in a tree-like fashion, and its hierarchical structure is stored in the root directory (~lager/LagerIV/cellib) as an OCT facet called "cellib.dir:directory:contents". This OCT facet is read in SCHEM to show the lists of library cells in a hierarchical fashion, and it can be recreated whenever there is an update in the library using a program called *symtree*. This way, we can dynamically maintain the library structure. Most commercial schematic entry systems instantiate a library cell by its name. This feature is also available in SCHEM, as well as two more on-line library calling routines mentioned above: palette and on-line listings. This on-line approach has proven to be desirable, because the user does not have to look at the library manual all the time. How these routines are implemented is described in the user's manual section.

4.5. The Extractor

Even though both the VEM schematic view and SMV follow the OCT symbolic policy[Bur88a], there are some differences in mapping according to their specifications, geometry being one(there is no geometry information in the SMV). In order to implement the vectorization of nets and terminals along with other features, an extractor program is written to convert a given schematic view of the design into the SMV. This function is explained in the following:

```
createSMV(&facet)
    delete geometry and unnecessary things from the facet;
    for every formal terminal {
      net name = terminal name;   /* if net name is not assigned */
      addQuote(TERM_EDGE, TERM_TYPE properties);
      addQuote(terminal name);   /* the LagerIV policy */
    }
    for every net {
      assign unique name if not assigned;
      addQuote(net name)
    }
    for every instance {
      if (terminal expander) {
        for every terminal {
          do necessary mapping into its net and original term;
            (attach BASE_INDEX & INCREMENT)
        }
        delete the instance;
      } else {
        create LagerIV properties(STRUCTURE_INSTANCE, ORDER);
        ReplaceInstance(from "symbol", to "SMV");
      }
    }
```

The OCT, oh(OCT helper) library routines are used to access the OCT database directly.

5. User's manual

This manual is a brief description of how to use SCHEM to generate the LagerIV SMV, and other essential information which the user needs to know in advance. This manual assumes that the user is already familiar with VEM.

Before running the application, VEM needs to know where and how to run it. This information must be included in the user's "vem.bindings" file, with the following two lines:

REMOTE schem localhost ~lager/LagerIV/bin/schem
schematic schem MENU Application 3 ALIAS

All the Lager paths and environment are assumed to be set correctly. The user needs to add the following lines in the lager file for this tool to find its path correctly:

```
(Schem
    ~lager/LagerIV/cellib
    ~lager/LagerIV/cellib/dpp/blocks
    ~lager/LagerIV/cellib/dpp.NEW/blocks
    ~lager/LagerIV/cellib/stdcell
    ~lager/LagerIV/cellib/TimLager
    ~lager/LagerIV/cellib/TimLager/dpram
    ~lager/LagerIV/cellib/TimLager/latch
    ~lager/LagerIV/cellib/TimLager/pla
    ~lager/LagerIV/cellib/TimLager/ram3T
    ~lager/LagerIV/cellib/termExpander
    etc.
)
```

For the TimLager cell library, the paths for all the blocks are separately added, since each block has its own leafcells. The lager is a startup file for LagerIV tools, which provides the UNIX pathnames for the directories required by various CAD tools in the LagerIV system. Refer to LagerIV Tools Users Manual for details. Furthermore, the name of the local machine(workstation) should be in the ".rhosts" file in order for RPC to work properly.

5.1. Design Flow

The following is a brief sequence of using this schematic tool.

(1) Execute **vem**

(2) Open a "cell:schematic" window from VEM

(3) Invoke **schem** by "schem: rpc-any"

(4) There are three ways of instantiating a cell from *getpart* pane:
   (a) Using **get-by-name(N)** if you remember the name of the cell
   (b) Using **list-cellib(l)** to choose one from an on-line list of library cells
   (c) Using a combination of **palette-lib(a)** and **instantiate(g)**

(5) Use the commands in *edit* pane to create properties specific to the design

(6) Use the commands in *net & term* panes to draw circuit interconnections

(7) Use the **create-SMV** command to create the structure_master view

(8) Use the **create-symbol** command to create the interface symbol

(9) Execute **DMoct** to create the SIV and generate the layout

5.2. Menu Description

Each SCHEM command has a key-binding for a faster and easier access. Currently, SCHEM has thirty customized menus in the six menu panes to assist the user in creating the SMV: *getpart*, *edit*, *term*, *net*, *show*, and *main*. Note that all the VEM commands are still available on the left button of the mouse. However, only basic VEM commands, such as *open-window*, *zoom*, and *pan*, are recommanded, since others are not compatible with the SMV construct. The *getpart* menu pane provides functions necessary to instantiate a cell either from the library or from the working directory. As explained above, the three ways of instantiating a cell provide the user the versatile library interface. Property editing utilities pertinent to the LagerIV policy are provided in the *edit* pane such **edit-formalPars**. The *term* menu pane provides options like creating/deleting formal terminals and editing terminal properties, while the *net* pane serves options for net editing such as **edit-netWidth**. The *show* menu provides all the routines showing the symbol view and the contents(**push-contents**) for the hierarchical design, as well as some basic properties showing. Finally, the *main* menu pane provides the schematic to the SMV converter and the symbol generator program. The usage of each function is now described in detail.

The character in parenthesis is the available RPC-key binding. For the LagerIV terminology used here, please refer to LagerIV Silicon Assembly System Manual.

get-by-name(N)

Given the part name, it finds the full UNIX path name for the cell using GetPath() available from the LagerIV library routines and instantiates it. The directory mapping is done from the lager file. Its identifier is *Schem* as explained above.

list-cellib(l)

This routine searches the hierarchical structure of the LagerIV cell library stored in "**cellib.dir:directory:contents**" and instantiates a cell of the user's choice.

palette-lib(a)

It has four palettes available now: stdcell, TimLager, dpp and a set of terminal expanders for the parameterized bus. A dialog box with the four selections is prompted, so that the user may choose whichever palette he wants.

instantiate(g)

Clicking a point in the VEM schematic for location, and moving the mouse into a cell of the user's choice, this routine finds the master of the cell and its SMV, and instantiate it. It also prompts for module parameters if there are any.

edit-formalPars(F)

It helps the user edit the parameters attached to the FORMAL_PARAMETERS bag. This routine first asks for the number of parameters, and then pops a customized dialog box according to the number the user specified.

edit-layoutGen(G)

This menu edits the LAYOUT_GENERATOR property. It does both creating and editing. If this command is executed for the first time, it pops a dialog box with the list of available layout generators; if not, it prompts another kind of dialog box with the specified LAYOUT_GENERATOR, so that the user may change the value.

edit-structProc(E)

This routine helps to edit interactively the structure processor available in LagerIV: dpp, plagen, Bds2stdcell, or makeFlatStdSIV.

edit-libraryCell(O)

It edits the library-cell property according to the LagerIV policy.

create-label(Y), delete-label(U)

These routines create and delete labels of the facet. They should soon become a part of the standard VEM menu.

create-term(T)

In VEM, the user can only creates a formal terminal over the physical implementation. This routine creates a formal terminal implementation without this constraint.

delete-term(d)

VEM also provides a standard *delete* command. But this SCHEM command also deletes a floating net, which VEM command fails to do.

edit-stdTerm(h)

Since every formal terminal for the stdcell layout generator has TERM_EDGE and TERM_RELATIVE_POSITION properties, it is customized to handle them.

edit-busProp(A)

Each node of the terminal expander is prompted for BASE_INDEX and INCREMENT properties for the vectorization of nets and terminals. The mouse should be on the terminal expander instance.

label-net(w)

This routine names a selected net specified by the user and shows its label. This is a shortcut of overcoming the lack of the displayable property in current OCT and VEM.

show-formalPars(H), show-layGen(J), show-structProp

These are customized routines to display the specific LagerIV properties into the VEM console window.

show-netProp(Q)

This routine displays all the net properties attached to a selected net, such as NETWIDTH and NET_TYPE.

show-symbol(v)

This routine shows a black box symbol representation of the current facet if it exists.

push-contents(V)

This routine allows the user to look at the contents of a schematic symbol at any level of the design hierarchy, except for the case of library cells.

create-SMV(C)

This RPC routine invokes the schematic-to-SMV translator, which was explained previously.

create-symbol(X)

This routine invokes the automatic symbol generator program.

smv-symbol(y)

This routine has the same function as **create-symbol**, but it also instructs the program to process from the SMV, not from the current schematic view. It is useful for creating library schematic symbols, which already have the SMVs.

clean-facet

This routine cleans mistakes in the current working facet. For instance, it deletes a floating net.

quit This routine exits the RPC application.

6. Conclusion

SCHEM, a parameterizable schematic entry system that uses automatic symbol generation, has been implemented to provide a graphical interface to the LagerIV silicon compiler system. This schematic tool serves the user with many merits of graphical interface, such as the visual understanding of the circuit in design and the ease with which changes may be performed interactively to an existing design. The hierarchical description of the circuit design is fully supported using the black box schematic symbol, which is produced automatically by the symbol generator program. One of the tutorial examples which has been generated out using SCHEM, a simple 4-bit microprocessor, is shown in Fig. 6 with the final physical layout.

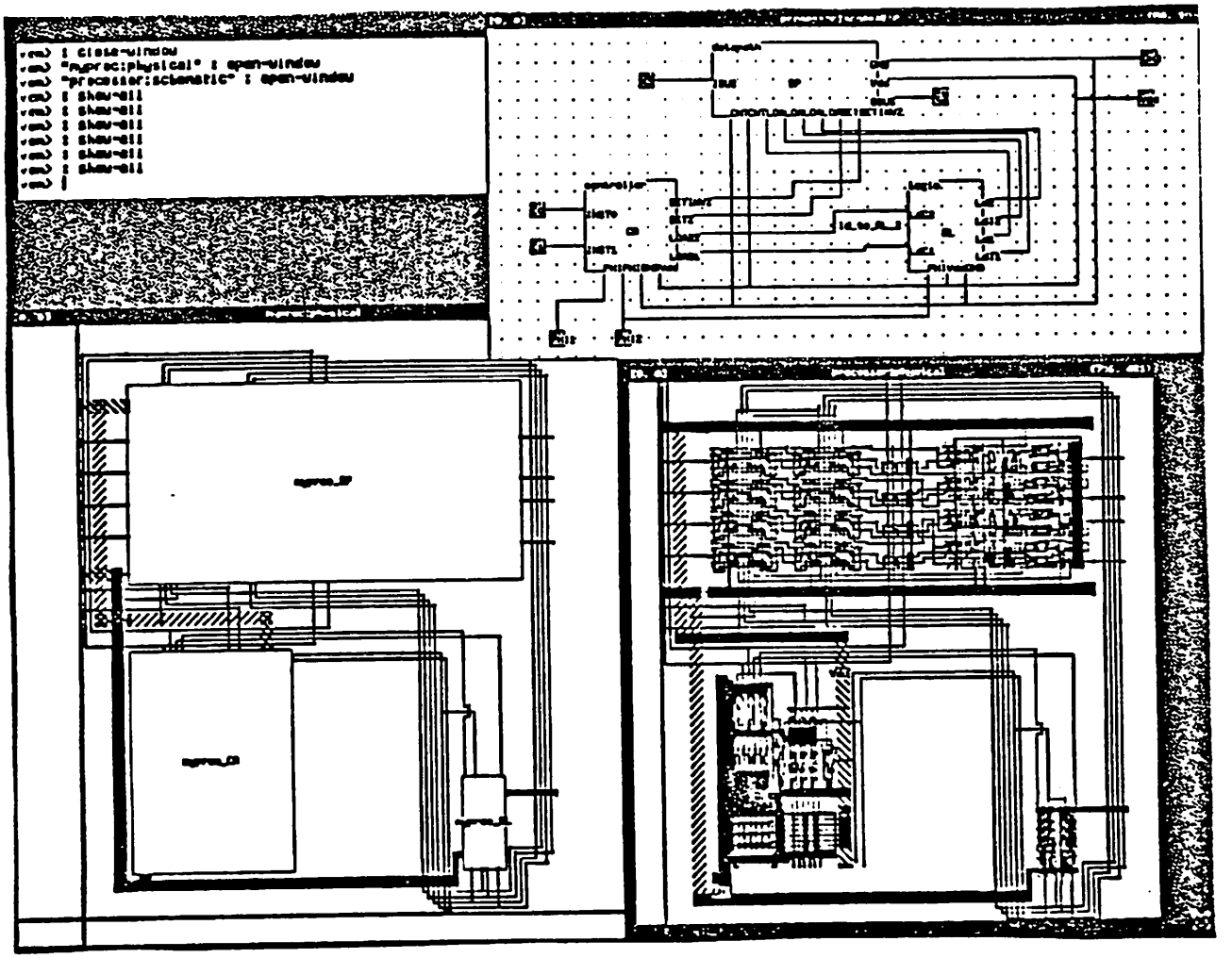

Fig. 6 an Example of Complete Layout

Future work includes completing the menu customization for an easier interface and interfacing into VEM7, which is built on the X11 window system and has an improved user interface, especially in the areas of editing and labeling. SCHEM is one of the first schematic capture systems that fully utilizes the advantages of the OCT/VEM/RPC environment. The idea of the parameterizability and hierarchical design, which SCHEM represents can be generalized to support many existing chip and printed circuit board design systems.

Acknowledgement

I would like thank Paul Tjahjadi at UCLA for providing useful library interface routines. Special thanks also to Rick Spickelmier and David Harrison for their helpful information, especially when OCT and VEM went through a major update. Finally, I thank Professor Jan Rabaey for his supervision and his useful advice.

References

Rab85a.
J. Rabaey, S. Pope, and R. Brodersen, "An Integrated Automated Layout Generation System for DSP Circuits," *IEEE Trans. on CAD*, vol. CAD-4, no. 3, July 1985.

Moo87a.
Peter Moore, David Harrison, Rick L. Spickelmier, and Richard Newton, "OCT, VEM and RPC," *Technical Report*, ERL, UC Berkeley, 1987.

Bur88a.
Jeff Burns and Rick Spickelmier, "Oct Symbolic View Specification," *Technical Report*, ERL, UC Berkeley, March 1988.

Har86a.
David Harrison, Peter Moore, Rick Spickelmier, and Richard Newton, "Data management and graphics editing in the Berkeley design environment," *ICCAD*, pp. 20-24, November 1986.

Spi87a.
Rick Spickelmier, "Remote Procedure Call Package," *Technical Report*, ERL, UC Berkeley, August 1987.

Jai88a.

Rajeev Jain, *LagerIV Silicon Assembly System Manual*, Release 1.0, ERL, UC Berkeley, 1988.

APPENDIX:

NAME

schem -- rpc version of schematic entry/capture system for creating structure_master view of LagerIV

DESCRIPTION

Schem is a remote-application program to enter schematics graphically in the LagerIV system. A user should first open a window of schematic view from VEM. **schem** can then be invoked by any of the standard ways of spawning rpc from VEM, such as the **rpc-any** command. If successful, VEM will display the message "SCHEM Operational" in the console window. The middle button on the mouse with the shift key pressed gives **schem** menus. Currently, it has six panes: *getpart*, *edit*, *net*, *term*, *show*, and *main*. All of the VEM schematic editing commands are also available.

OPTIONS

The following options are available on the application-menu panes that schem registers. The character in the parenthesis is the RPC key binding for each command:

get-by-name(N)
instantiate a cell by typing its name

palette-lib(a)
open one of the four palette windows(stdcell, dpp, TimLager, termExpander)

list-cellib(l)
show the list of LagerIV library cells in a hierarchical fashion and instantiate the chosen cell

instantiate(g)
instantiate the cell under the cursor

edit-formalPars(F)
edit the FORMAL_PARAMETERS bag

edit-layoutGen(G)
edit the LAYOUT_GENERATOR property

edit-structProc(E)
edit the STRUCTURE_PROCESSOR property

edit-libraryCell(O)
edit the LIBRARY_CELL property

create-label(Y)
create label into the facet

delete-label(U)
delete label from the facet by its name

create-term(T)
create a formal terminal

delete-term(d)
delete a formal terminal

edit-stdTerm(h)
edit the TERM_EDGE and TERM_RELATIVE_POSITION properties for stdcell design

edit-termType
edit the TERMTYPE property of a terminal

edit-busProp(h)
edit the BASE_INDEX and INCREMENT properties at each node of the termExpander

edit-netWidth
edit the NETWIDTH property of a selected net

edit-netType
edit the NETTYPE property of a selected net

label-net(w)
name a net with its label shown

show-formalPars(H)
show the contents of the FORMAL_PARAMETERS bag

show-layoutGen(J)
show the LAYOUT_GENERATOR property

show-strctProp
show the STRUCTURE_PROCESSOR property

show-netProp(Q)
show the properties attached to a selected net

show-symbol(v)
show the schematic symbol of the current facet

push-contents(v)
show the contents of the schematic symbol under the cursor

create-SMV(C)
create the structure_master view

create-symbol(X)
generate the interface symbol for the cell

clean-facet
clean the current facet. This command deletes floating nets, etc. create-SMV automatically invokes this before doing any processing

quit(q)
exit the application

The following is a brief sequence of using this schematic tool to generate the SMV:

(1) Execute **vem**

(2) Open a "cell:schematic" window from VEM

(3) Invoke **schem** by "schem: rpc-any"

(4) There are three ways of instantiating a cell from *getpart* pane:
(a) Using **get-by-name(N)** if you remember the name of the cell
(b) Using **list-cellib(l)** to choose one from on-line list of library cells
(c) Using a combination of **palette-lib(a)** and **instantiate(g)**

(5) Use the commands in *edit* pane to create properties specific to the design

(6) Use the commands in *net & term* panes to draw circuit interconnections

(7) Use the **create-SMV** command to create the structure_master view

(8) Use the **create-symbol** command to create the interface symbol

(9) Execute **DMoct** to create the SIV and generate the layout

A user needs to add the following lines in his/her "lager" file:

```
(Schem
        "lager/LagerIV/cellib
        "lager/LagerIV/cellib/dpp/blocks
        "lager/LagerIV/cellib/dpp.NEW/blocks
        "lager/LagerIV/cellib/stdcell
        "lager/LagerIV/cellib/TimLager
```

~lager/LagerIV/cellib/TimLager/dpram
~lager/LagerIV/cellib/TimLager/latch
~lager/LagerIV/cellib/TimLager/pla
~lager/LagerIV/cellib/TimLager/ram3T
~lager/LagerIV/cellib/termExpander
etc.
)

Also, the name of the local machine(workstation) should be in the ".rhosts" file in order for RPC to work properly.

Please note that Vdd and GND creation is implicit in **stdcell** layout cells. This is done automatically in the SMV creation process.

SEE ALSO

vem(1), DMoct(1)

EXAMPLES

See ~minj/schem/examples for some examples.

BUGS

New commands are still being added in SCHEM. The LagerIV system needs to be updated to OCT2 for the SMV, created by SCHEM, to be recognized by DMoct. The future version will support VEM7/X11 with a better user interface.

AUTHORS

Jonathan Min (minj@zabriskie.Berkeley.EDU)
Paul Tjahjadi (tjahjadi@zabriskie.Berkeley.EDU)