VIS Programmer's Manual

Thomas Shiple

The VIS Group

Adnan Aziz[1]
Robert Brayton[1]
Stephen Edwards[1]
Gary Hachtel[2]
Sunil Khatri[1]
Yuji Kukimoto[1]
Abelardo Pardo[2]
Shaz Qadeer[1]
Rajeev Ranjan[1]
Alberto Sangiovanni-Vincentelli[1]
Shaker Sarwary[3]
Thomas Shiple[1]
Fabio Somenzi[2]
Gitanjali Swamy[1]
Tiziano Villa[1]

[1]University of California, Berkeley
[2]University of Colorado, Boulder
[3]Now at Lattice Semiconductor

`vis@ic.eecs.berkeley.edu`
May 10, 1996

1 Introduction

This manual provides a brief overview of the architecture of VIS. The first section looks at VIS as a whole, and subsequent sections cover the major components of VIS.

VIS was designed to be modular and lightweight. By understanding the architecture of the system, future VIS developers can work to maintain these attributes.

2 VIS

VIS is partitioned into three main components:

1. VIS-F — The front end. It provides the ability to read and write BLIF-MV files, and supports a hierarchical data structure mimicking the constructs of BLIF-MV.
2. VIS-V — The verification system. This provides facilities for combinational and sequential equivalence checking, fair CTL model checking, and cycle-based simulation.
3. VIS-S — The synthesis system. This provides state minimization, variable encoding, and hierarchical restructuring capabilities.

Figure 1 is a block diagram showing how the three components interact. The packages that constitute each component are listed along with edges denoting dependencies among the packages. `glu` is the Generic Libraries Utility, which contains utility packages such as `array`, `list`, and `bdd`. Note that

- VIS-F does not depend on VIS-V or VIS-S, and
- VIS-V and VIS-S are independent.

The first point allows VIS to be easily compiled leaving out VIS-V, VIS-S, or both, to produce an executable containing a subset of the capabilities. The second point forces communication between verification and synthesis to occur via the front end, rather than directly.

The division of VIS into the three components is not reflected in the directory structure of the source code. Instead, all packages are kept within a single directory named `src`.

3 VIS-F: Front End

VIS-F is the front end. It provides an in-memory representation of BLIF-MV. This hierarchical representation can be traversed and manipulated. VIS-F consists of the following packages:

- `vm` — Contains the `main()` function, and provides the compilation date, version number, and the location of the VIS library.[1]
- `cmd` — The interactive command interface. Provides a global table to store values for user-settable variables (e.g., the value `autoexec`). Also provides the system level commands like `help`, `alias`, and `set`.[2]

[1] Largely borrowed from the `main` package of SIS.
[2] Largely borrowed from the `command` package from SIS.

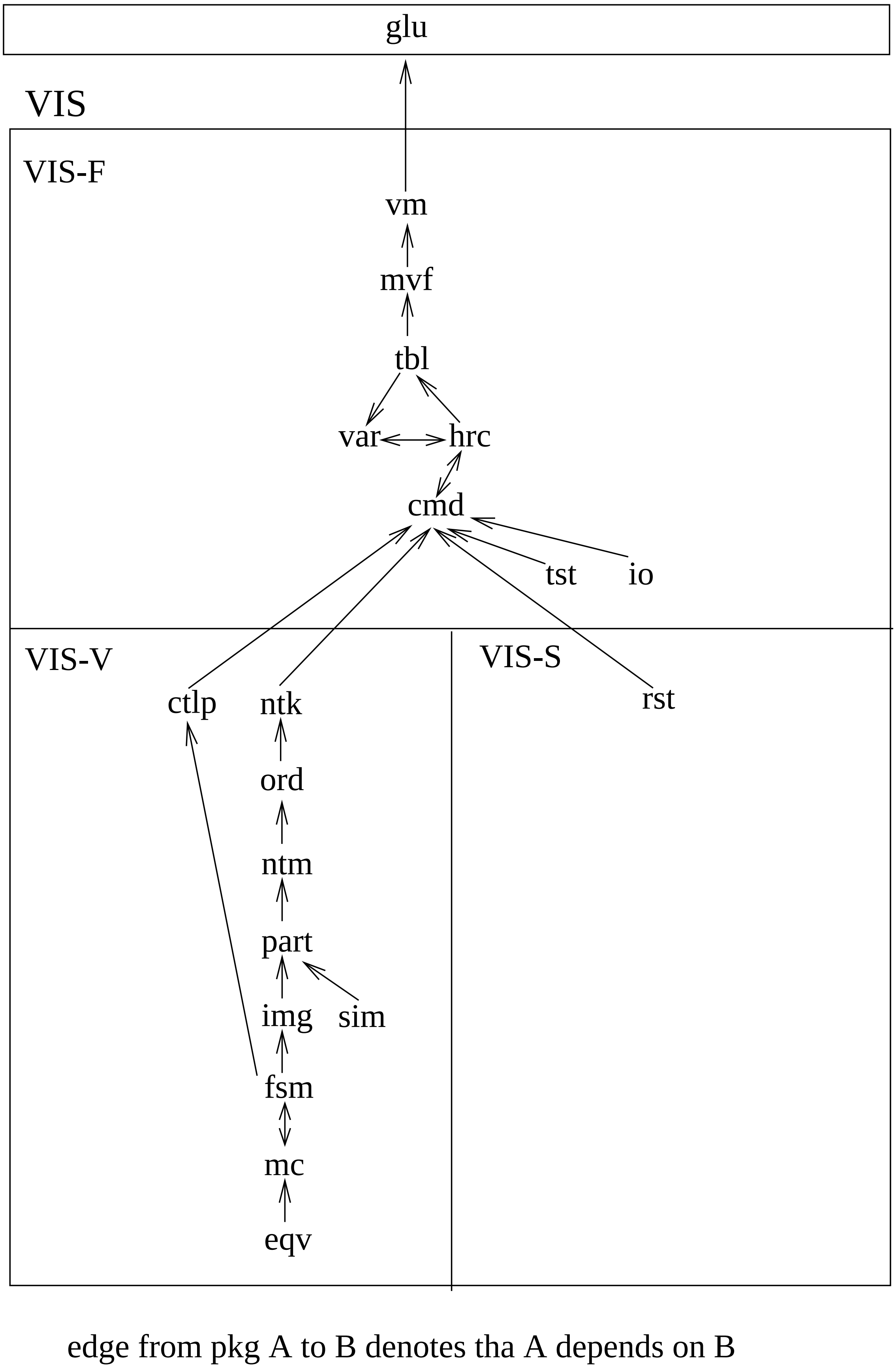

Figure 1: Components and packages of VIS. An edge from package A to B denotes that A depends on B (edges implied by transitivity are not shown).

- `mvf` — A data structure to represent multi-valued input, multi-valued output functions, based on BDDs.
- `tbl` — A data structure to represent multi-valued relations, in particular the `.table` construct in BLIF-MV.
- `var` — A data structure to represent multi-valued variables, in particular the `.mv` construct in BLIF-MV.
- `hrc` — Data structures to represent a hierarchical design, in particular the `.model`, `.subckt`, and `.latch` constructs in BLIF-MV.
- `io` — Routines to read and write BLIF-MV and BLIF files.
- `tst` — A package template that can be used as the starting point for the creation of new packages.

When a BLIF-MV file is parsed, a directed acyclic graph of models is created by the `io` package, corresponding to the hierarchy given in the file. The DAG is then transformed by `io` into a tree by creating separate nodes for each instantiation of a model. Traversal and manipulation of the hierarchy takes place on the tree, and not the DAG, using routines provided by `hrc`.

The `Hrc_Node_t` data structure provides a lookup table for applications (e.g., VIS-V and VIS-S) to store data associated with a node in the hierarchy. In this manner, VIS-F can remain independent of VIS-V and VIS-S.

4 VIS-V: Verification

VIS-V provides analysis capabilities for designs. From any node in the hierarchy (the *current* node), executing the command `flatten_hierarchy` causes a flattened *network* to be created, representing everything from the current node down to the leaves of the hierarchy. Having a flattened representation in which all combinational "gates" and latches exist in a single network allows for the global analysis of that part of the design encompassed by the current node of the hierarchy. The packages of VIS-V are:

- `ntk` — A directed graph representation, where the vertices are "gates," inputs and latches. Combinational cycles are not precluded by the network data structure, but many of the packages assume the absence of combinational cycles.
- `ord` — Routines to order the MDD variables of a network, based on the structure of the network. Also provides an interface to dynamic ordering of variables.
- `ntm` — A routine to build the `Mvf_Function_t`s of the roots of an arbitrary region of a network, in terms of the leaves of the region. The leaves can be treated as variables or as specific constants.
- `part` — Routines to build an MVF representation of a network. The MVFs are stored at the vertices of a DAG, where the sinks correspond to the combinational outputs of the network, and the sources to the combinational inputs. In general, intermediate vertices can be introduced to control the size of the MVFs.
- `sim` — A cycle-based network simulator. Simulation is performed by evaluating the MVFs provided by the `part` package. Simulation vectors can be provided by the application, or random simulation can be performed.

- `img` — Generic routines for performing forward and backward image computation. The routines work off the graph of MVFs provided by the `part` package, and have no direct knowledge of the `ntk` or `fsm` packages. Since this is an active area of research, a generic interface has been designed to easily allow the addition of new computation methods.

- `fsm` — An abstraction of a network. The FSM does not actually store the next state functions of the FSM — these are provided by `part`. It does store the vectors of present state and next state variables, reachability information, fairness constraints-related information, and image computation information.

- `mc` — A fair CTL model checker and debugger for FSMs.

- `eqv` — Routines for performing combinational equivalence between regions of two networks, and for performing sequential equivalence between two FSMs.

- `ctlp` — A CTL parser.

The nodes of a network have a single output. The function of a combinational node of a network is represented by a `Tbl_Table_t`. A k-output table in the hierarchy is represented by k combinational nodes in the corresponding network, where each of the k nodes points to the same table, but are distinguished by which output column they represent. This splitting is done by the flattening routine in the `ntk` package.

Throughout VIS-V, it is assumed that the combinational outputs are completely specified and deterministic. Non-determinism is introduced via *pseudo-inputs*. A pseudo-input is like a non-deterministic constant that can update its value on each clock cycle. See the documentation for the `ntk` package for more information on pseudo-inputs.

Because the emphasis of VIS-V is on analysis, the ability to modify the network data structure is not provided. It is assumed that once a network is created from the hierarchy, the network will be unchanged until it is destroyed.

Notice that the `ntk` package is independent of all other packages in VIS-V. This independence is maintained by allowing applications (e.g., `fsm`, `part`) to store information associated with a network in a lookup table. It is important to maintain this independence so that the `ntk` data structures do not become cluttered.

5 VIS-S: Synthesis

Only one package has been written thus far for this component. It is anticipated that packages will be added to support state minimization, variable encoding, and other operations. However, note that VIS-F already allows a BLIF file to be written, which can be massaged by the sequential synthesis system SIS and read back into the hierarchy. The packages of VIS-S are:

- `rst` — Routines for restructuring the hierarchy.

6 Possible Improvements

1. The `mvf` package could be located in the Generic Libraries Utility.

2. One existing complication in removing the restriction that networks can't be modified is that the `Tbl_Table_t` and `Var_Variable_t` data within a

network are owned by (and hence freed by) the hierarchy manager (**`hrc`** package). This could be resolved by creating a global manager for tables and a global manager for variables.