The VIS Engineering Manual

Stephen Edwards Gitanjali Swamy

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

Contents

Chapter 1

Introduction

This document describes a set of conventions[1] for organizing and writing C code in the VIS system[2]. The conventions concern the naming of functions, macros, and variables; the style of documentation within the code; and the organization and naming of source files.

The reasons for adopting these conventions are

- Writability: Design decisions are simplified.
- Maintainability: Consistency makes the code comprehensible to other programmers.
- Predictability: Conventions about names and comments make it easy to write scripts that automatically extract information (documentation and prototype extraction).

In this document, we will discuss how code within VIS is organized, how identifiers (functions, macros, structures, etc.) within VIS are named, and finally how code within VIS is maintained.

Chapter 2 describes how C code within VIS is divided into packages, a group of files centered around either a data structure or a function. It also introduces the *structured comment*, a C comment formatted to allow the automatic extraction of documentation.

Chapter 3 discusses naming conventions for functions, macros, structures, and variables. These conventions make the type, scope, and function of identifiers evident from their names. Chapter 4 illustrates these conventions with a simple package.

Chapter 5 describes maintenance programs for this environment, and Chapter 6 describes how to compile and how to create a new package under VIS.

---

[1] Many of these conventions come from John Ousterhout's *Tcl/Tk Engineering Manual*, available with the Tcl/Tk distribution from `ftp.cs.berkeley.edu`. The function naming conventions and the automatic prototype extractor were also inspired by the ideas of Herve Touati, Jean Christophe Madre, and Olivier Coudert. The documentation extractor for Sun's Java language is similar to ours (`http://java.sun.com`).

[2] Verification Interacting with Synthesis

Chapter 2

Packages and File Structure

2.1 Packages

Code in VIS is divided into packages. A package is a group of source (`.c`) and header files (`.h`) in a subdirectory centered around either a function (e.g., the BDD variable ordering package "`ord`"), or a data structure (e.g., the network package "`ntk`"). Each package has a short two- to five-letter name, called the short package name, placed in front of all filenames and external identifiers related to the package. This name may appear in all lowercase, or with its first letter capitalized, written as *package* and *Package* respectively.

Each package should contain two header files. The external header file, named *package*`.h`, defines features visible from outside the package. The internal header file, named *package*`Int.h`, defines features used in multiple files inside the package, but not outside. If necessary, a third header file, named *package*`Port.h`, can be included. This should contain definitions for the package that hide differences between systems. No additional header files should be used—everything should go into the two main headers.

The names of C source files begin with the short package name followed by a series of English words (or abbreviations) with their first letters capitalized. No filename can be longer than fifteen characters. (This is a limitation imposed by the library archive program `ar`.)

Listed below are the names of files in an example package called `rng`. It has an external header file, an internal header file, a header file for portability, and two `.c` files. Other packages may have more `.c` files, but should not have more `.h` files.

Name	Contains
`rng.h`	Externally-visible functions, etc.
`rngInt.h`	Functions, etc., internal to the package
`rngPort.h`	`#defines`, etc., that hide system differences
`rngUtil.c`	Utility functions
`rngIntUtil.c`	Internal utility functions

2.2 C Header Files

Both the internal and external headers follow the same structure:

1. A `CHeaderFile` structured comment describing the contents of the file, described in Section 2.4.1. The external header file's comment should be directed toward programmers who will use the package; the internal header file's comment should be directed toward maintainers. Both should be self-contained.

2. `#ifdef` and `#define` statements that prevent this from being `#included` twice. In the external header file, the name should be an underscore followed by the short name of the package in all caps, e.g., `_NTK`. In the internal header file, the name should be an underscore followed by the short name of the package followed by "`INT`", e.g., `_NTKINT`.

3. In the internal header file, `#include` the external header files from other packages that are *directly* needed for internal or static code, but are not needed for external code. In addition, *package*`Int.h` should include *package*`.h`.

   package`.h` should include those external header files that are *directly* needed for the structures and functions that *package* exports. In this manner, any package that includes *package* automatically gets all of the packages that are needed to compile *package*.

4. Declarations of non-functions, each type in a separate section with a leading comment.

5. An automatically-generated section surrounded by `AutomaticStart` and `AutomaticEnd` comments. Anything between these comments is removed by the automatic prototype extractor, so the programmer should put nothing here.

6. The `#endif` directive for the leading `#ifdef`.

A template C header file that includes all the required sections and comments is shown below.

```
/**CHeaderFile*****************************************************************
  FileName    [ required ]
  PackageName [ required ]
  Synopsis    [ required ]
  Description [ optional ]
  SeeAlso     [ optional ]
  Author      [ optional ]
  Copyright   [ required ]
  Revision    [ $Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $ ]
******************************************************************************/
#ifndef _
#define _

/*---------------------------------------------------------------------------*/
/* Constant declarations                                                     */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Type declarations                                                         */
/*---------------------------------------------------------------------------*/
```

```
/*---------------------------------------------------------------------------*/
/* Stucture declarations                                                     */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Variable declarations                                                     */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Macro declarations                                                        */
/*---------------------------------------------------------------------------*/

/**AutomaticStart*************************************************************/

/*---------------------------------------------------------------------------*/
/* Function prototypes                                                       */
/*---------------------------------------------------------------------------*/

/**AutomaticEnd***************************************************************/

#endif /* _ */
```

2.3 C Source Files

A C source file contains the following:

1. A `CFile` structured comment describing the contents of the file, described in Section 2.4.2.
2. `#include "`*package*`Int.h"`.
3. An RCS string, used by the revision control system described in Chapter 5.
4. Declarations and definitions of non-functions, each type in a separate section with a leading comment.
5. An automatically-generated section surrounded by `AutomaticStart` and `AutomaticEnd` comments. Anything between these comments is removed by the automatic prototype extractor, so the programmer should put nothing here.
6. Three sections, each with a leading comment, separated into definitions for
   (a) Functions visible outside the package
   (b) Functions visible to all files within the package only
   (c) Functions visible to the file only

A template C source file that includes all the required sections and comments is shown below.

```
/**CFile***********************************************************************
  FileName    [ required ]
  PackageName [ required ]
  Synopsis    [ required ]
  Description [ optional ]
  SeeAlso     [ optional ]
  Author      [ optional ]
```

```
  Copyright   [ required ]
******************************************************************************/

#include "Int.h"

static char rcsid[] = "$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $";
USE(rcsid);

/*---------------------------------------------------------------------------*/
/* Stucture declarations                                                     */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Type declarations                                                         */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Variable declarations                                                     */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Macro declarations                                                        */
/*---------------------------------------------------------------------------*/

/**AutomaticStart*************************************************************/

/**AutomaticEnd***************************************************************/

/*---------------------------------------------------------------------------*/
/* Definition of exported functions                                          */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Definition of internal functions                                          */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/* Definition of static functions                                            */
/*---------------------------------------------------------------------------*/
```

2.4 Structured Comments

Documentation for packages, functions, macros etc. is embedded within C source and header files in "structured comments," which can be understood by programs. For example, the `extdoc` program described in Section 5.2 can extract this documentation and format it for a text file or in HTML format suitable for the World Wide Web. The `extproto` program described in Section 5.1 also uses these comments to extract function declarations.

Any definition (e.g., functions, macros, structures) must be preceded by a structured comment that indicates information about its functionality, use, etc. We suggest you write this documentation for your package before you write its code. This ensures the documentation gets written and allows you to see the package's complete interface.

A structured comment begins with a line beginning with "`/**`" (i.e., with no leading whitespace). The type of the comment (e.g., `CFile`, `Function`) is an alphanumeric string on the first line that may be embedded in asterisks or whitespace. A structured comment ends with a line containing "`*/`".

Between the start and end lines are field-value pairs separated by whitespace. A field is an alphanumeric string and a value is a square bracket-enclosed string that may include newlines. A value may not include a close-bracket unless it is escaped with a backslash, i.e., `\]`. The order of field-value pairs in a structured comment is irrelevant, but they must appear in pairs. A field-value pair named `Comment` is ignored by all automatic extraction tools, allowing comments within structured comments.

Certain values, such as the `Synopsis` and `Description` fields, may include HTML[1] directives. These include `<pre>` and `</pre>`, which begin and end a section of preformatted text, i.e., that will be displayed with spaces and newlines intact, and `<p>`, which indicates a paragraph break.

2.4.1 `CHeaderFile`

The `CHeaderFile` structured comment at the beginning of a header file contains the following fields, some of which are optional:

`FileName`	required	The name of the header file
`Package`	required	The short name of the package, all lowercase
`Synopsis`	required	A one-line summary of the package
`Description`	optional	A more detailed description of the package, possibly multiple paragraphs. In the external header file, this should be directed toward users of the package. In the internal header file, this should be for maintainers of the package. This should be self-contained in both cases.
`SeeAlso`	optional	A space-separated list of names of things that may be of interest (e.g., packages, files, etc.)
`Author`	optional	The author or authors of the package
`Copyright`	required	A copyright notice for the file
`Revision`	required	The string `$ Id: vis\_eng.tex,v 1.11 1995/07/08 20:32:28 sedwards Exp gms $`, used by the revision control system.

An example `CHeaderFile` comment is shown below.

```
/**CHeaderFile*****************************************************************
  FileName    [ rng.h ]
  PackageName [ rng ]
  Synopsis    [ Manipulation of integer ranges, e.g., 5-7. ]
  Description [ Routines for creating, adding to, querying, and
                deleting ranges. ]
  SeeAlso     [ rngInt.h ]
  Author      [ Gitanjali Swamy ]
  Copyright   [ Copyright (c) 1994-1996 The Regents of the Univ. of California.
                All rights reserved. ]
  Revision    [$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $]
******************************************************************************/
```

[1]HTML: Hyper Text Markup Language, used on the World-Wide Web. A complete description of these directives may be found at
`http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html`.

2.4.2 CFile

The `CFile` structured comment at the beginning of a source file contains the following fields, some of which are optional:

`FileName`	required	The name of the source file
`Package`	required	The short name of the package, all lowercase
`Synopsis`	required	A one-line summary of the file's contents
`Description`	optional	A more detailed description of the file, possibly multiple paragraphs. It should be self-contained.
`SeeAlso`	optional	A space-separated list of names of things that may be of interest (e.g., packages, files, etc.)
`Author`	optional	The author or authors of the file
`Copyright`	required	A copyright notice for the file

An example `CFile` comment is shown below.

```
/**CFile*********************************************************************
  FileName    [ rngUtil.c ]
  PackageName [ rng ]
  Synopsis    [ Memory and read/write utilities for the Range package ]
  Description [ Contains routines for creating and deleting ranges,
                and asking questions of them ]
  SeeAlso     [ rngUtilInt.c ]
  Author      [ Gitanjali Swamy ]
  Copyright   [ Copyright (c) 1994-1996 The Regents of the Univ. of California.
                All rights reserved. ]
******************************************************************************/
```

2.4.3 Function

The fields in the `Function` structured comment are

`Synopsis`	required	A one-line synopsis of the function
`Description`	optional	A longer description of the function: its parameters, its return type, and how it is computed. This should be self-contained.
`SideEffects`	required	A description of any side effects (e.g., modification of global variables). If there are none, include an empty string: "`[]`".
`SeeAlso`	optional	A space-separated list of the names of related functions, files, packages, etc.
`CommandName`	optional	The name of the command that invokes this function. Omit when the function is not a command.
`CommandSynopsis`	optional	A one-sentence synopsis of the command.
`CommandArguments`	optional	The command-line arguments recognized or required by the command.
`CommandDescription`	optional	A full description of the command.

To document a function's parameters and return type, insert standard `/* */`-enclosed C comments after each parameter and after the return type. Parameter comments should follow the parameter name, before the comma or close-parenthesis. A return type comment should be between the return type and the function name. All of these comments are optional.

If the function can be invoked directly from the command line, it should include the `Command` fields. The sentence in the `CommandSynopsis` field should be an imperative command, e.g., "read a network" instead of "reads a network." Omit a trailing period. The `CommandArguments` field should have symbolic names enclosed in angle brackets (e.g., `<list>`), and optional arguments enclosed in escaped square brackets (e.g., `\[ -v \]`). The `CommandDescription` field should fully document the command, and may use arbitrary HTML constructs.

An example `Function` comment with partial definition is shown below.

```
/**Function*******************************************************************
  Synopsis    [ Check if two ranges overlap ]
  Description [ If any integer is common to both of the given ranges,
                return TRUE, otherwise return FALSE. ]
  SideEffects []
  SeeAlso     [ Rng_RangeOrWithRange ]
******************************************************************************/
static boolean /* TRUE if the ranges overlap */
RangeOverlap(
  Rng_Range_t * range1 /* first range */,
  Rng_Range_t * range2 /* second range */)
{ /* ... */ }
```

2.4.4 `Macro`

The `Macro` fields in a structured comment are identical to those for `Function`. It is intended that macros are indistinguishable from functions.

To document the arguments in a macro, insert standard C comments before each argument name to give the type, and after each argument name (but before a comma or close-parenthesis) to comment the argument. These comments are optional.

An example `Macro` comment with a partial definition is shown below.

```
/**Macro**********************************************************************
  Synopsis    [ Return the lower number in a range ]
  SideEffects []
  SeeAlso     [ Rng_RangeSetEnd ]
******************************************************************************/
#define /* int */ Rng_RangeReadEnd(                 \
  /* Rng_Range_t * */ range /* argument comment */  \
)                                                   \
(((range->begin) =< RNG_MAX)? (range->begin): RNG_MAX)
```

2.4.5 `Struct` and `Variable`

`Struct` and `Variable` structured comments have identical fields:

`Synopsis`	required	A one-line synopsis
`Description`	optional	A longer description. This should be self-contained.
`SeeAlso`	optional	A space-separated list of the names of related functions, files, packages, etc.

To comment the fields in a structure definition, place a C comment after each field definition, after the semicolon. These comments are optional.

Example `Struct` and `Variable` comments are shown below.

```
/**Struct**************************************************************
  Synopsis      [ Represents an integer range ]
  Description   [ Uses two integers to represent the range.
                  Under normal circumstances, begin <= end. ]
  SeeAlso       [ Rng_RangeAlloc Rng_RangeFree ]
```

```
******************************************************************/
typedef struct RngRangeStruct {
    int begin; /* The lower limit */
    int end;   /* The upper limit */
} Rng_Range_t;

/**Variable********************************************************
  Synopsis      [ The number of ranges current in existence ]
  Description   [ Whenever Rng_RangeAlloc is called, this is incremented.
                  Whenever Rng_RangeFree is called, this is
                  decremented.  This should always be positive

  SeeAlso       [ Rng_RangeAlloc Rng_RangeFree ]
******************************************************************/
int Rng_numRanges;
```

2.5 Low-Level Coding Conventions

To make code in the VIS environment look consistent, we have the following conventions about its appearance:

- **Lines are eighty characters wide.** Anything wider is difficult to read and print. Insert line breaks as necessary to ensure this.
- **Indents are two spaces.** This is smaller than other systems, but reduces the chance of nested instructions marching off the right side of the page.
- **Comments within code occupy full lines.** Comments tacked on to the right of statements can be hard to see and make it difficult to modify the code. Comments to the right of variable declarations, structure members, and arguments in function definitions are fine.
- **Open-braces go at the end of lines.** Close-braces should come at the beginning of a following line. The braces enclosing a function definition are the one exception to this rule.

  Always use curly braces in compound statements, even if they enclose only one statement. This makes it easier to correctly add statements to the block, and simplifies setting breakpoints in a debugger.

The following fragment illustrates these conventions.

```
void
Ntk_NodeDeclareAsCombinational(
  Ntk_Node_t * node,
  Tbl_Table_t * table,
  array_t * inputNames /* array of char */,
  int  outputIndex)
{

  /*
   * If the output column of a table can take only
   * a single value, set the constant flag.
   */

  if ( Tbl_TableTestIsConstant(table, outputIndex) ) {
    node->constant = 1;
  }

  /*
```

```
   * Print the node's name, MDD id, type, and attributes.
   */

  (void) fprintf(fp, "%s: mdd=%d, %s;%s%s%s%s%s%s\n",
                  Ntk_NodeReadName(node),
                 Ntk_NodeReadMddId(node),
                 typeString,
                 (Ntk_NodeTestIsPrimaryOutput(node) ? " output" : ""),
                 (Ntk_NodeTestIsConstant(node) ? " constant" : ""),
                 (Ntk_NodeTestIsLatchDataInput(node) ?
                  " data-input" : ""),
                 (Ntk_NodeTestIsLatchInitialInput(node) ?
                  " initial-input" : ""),              ,
                 (Ntk_NodeTestIsCombInput(node) ? " comb-input" : ""),
                 (Ntk_NodeTestIsCombOutput(node) ? " comb-output" : "")
                 );
}
```

Chapter 3

Naming Conventions

This chapter discusses the conventions for naming functions, macros, structures, and variables within the VIS environment. The objective of these conventions is for a name to be canonical and to completely identify its type and scope. Adhering to these conventions is necessary for the automatic prototype generation and documentation tools to work, described in Chapter 5.

When names are longer than one English word, the words are catenated and their first letters capitalized, e.g., `ThisIsALongExample`. This is shorter than other styles, and is at least as readable.

The environment defines three scopes, each of which has its own naming convention:

- External: features visible outside the package. Names are prefixed with "*package*_".
- Internal: features not visible outside, but visible throughout the package. Names are prefixed with "*package*".
- Local: features visible only within a single file or function. Names have no package prefix.

Functions and macros, variables, and structures each have their own naming convention:

- Functions/Macros: The package name and the first word of the rest of the name are both capitalized.

  The words in a function name are arranged in an English sentence-like order:

  1. First word: Name of the object being operated on: the subject of the sentence (e.g., `Range`, `Network`).
  2. Second word: The action being performed on the object: the verb of the sentence (e.g., `Read`, `Obtain`, `Alloc`). `Obtain` is used for functions that return a pointer to a copy of a data structure that must later be freed by the caller. `Read` is used for all other cases. `Get` should not be used.
  3. Third and successive words: Modifiers: the object of the sentence (e.g., `States`, `RightChild`, `BddIdArrayFromMddId`).

- Variables: The package name is lowercase. The first word of the rest of name is lowercase except for internal variables.
- Structures: Type definitions for these have the package name and the first word is capitalized, and the name has a `_t` suffix.
- Constants: All letters are capitalized; all words are separated by underlines ("_").
- Enumerations: Type definitions for these have the package name and the first word capitalized, and the name as a `_c` suffix.

The table below gives examples of names.

	External	Internal	Local
Functions, Macros	`Rng_RangeReadLower`	`RngRangeReadLower`	`ReadLower`
Variables	`rng_numRanges`	`rngNumRanges`	`numRanges`
Structures	`Rng_IntRange_t`	`RngIntRange_t`	`IntRange_t`
Enumerated types	`Rng_FastMethod_c`	`RngFastMethod_c`	`FastMethod_c`
Constants	`RNG_MAX_INT`	`RNGMAX_INT`	`MAX_INT`

Chapter 4

An Example Package

This chapter contains an example package intended to clarify the conventions. This is a package called "rng" that manipulates a data structure describing an integer range, i.e., sets of the form $\{x : L \leq x \leq U\}$ for integers L and U.

This small, contrived example contains 4 files: two header files, rng.h and rngInt.h; and two source files, rngUtil.c, and rngIntUtil.c.

4.1 The External Header File rng.h

```
/**CHeaderFile*****************************************************************
  FileName    [rng.h]
  PackageName [rng]
  Synopsis    [This package deals with functions that are used to manipulate
  the range (Rng_Range_t) data structure.]

  Description [ The basic data structure in the range package is an
  integer range. The range is designated as {L,U}, where L is the lower limit of
  the range, and U is the upper limit of the range, and
  includes all integers x, L =< x =< U).]
  SeeAlso     [tbl]
  Author      [Gitanjali Swamy]

  Copyright   [Copyright (c) 1994-1996 The Regents of the Univ. of California.
  All rights reserved.]
  Revision    [$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $]

******************************************************************************/
#ifndef _RNG
#define _RNG

/*---------------------------------------------------------------------------*/
/* Constant declarations                                                     */
/*---------------------------------------------------------------------------*/
int RNG_MIN = 0;
int RNG_MAX = 5000;

/*---------------------------------------------------------------------------*/
/* Type declarations                                                         */
/*---------------------------------------------------------------------------*/
typedef struct RngRangeStruct Rng_Range_t;

/*---------------------------------------------------------------------------*/
/* Variable declarations                                                     */
/*---------------------------------------------------------------------------*/
extern int Rng_NumRanges;

/*---------------------------------------------------------------------------*/
/* Macro declarations                                                        */
```

```
/*---------------------------------------------------------------------------*/

/**Macro***********************************************************************
  Synopsis     [Return the beginning of the range.]
  Description  [Given a range {L,U}, this returns the beginning value
               L of the range.]
  SideEffects []
  SeeAlso      [Rng_RangeReadEnd]
******************************************************************************/
#define Rng_RangeReadBegin(range)\
(((range->begin) >= RNG_MIN)? (range->begin): RNG_MIN)

/**Macro***********************************************************************
  Synopsis     [return the end of the range.]
  Description  [Given a range {L,U}, this returns the ending value
               U of the range.]
  SideEffects []
  SeeAlso      [Rng_RangeReadBegin]
******************************************************************************/
#define Rng_RangeReadEnd(range)\
(((range->begin) =< RNG_MAX)? (range->begin): RNG_MAX)

/**Macro***********************************************************************
  Synopsis     [Set the beginning of the range.]
  Description  [Given a range and a value L, this sets the beginning value
               L of the range.]
  SideEffects  [There original beginning value of the range is lost.]
  SeeAlso      [Rng_RangeSetEnd]
******************************************************************************/
#define Rng_RangeSetBegin(range,val)\
((val >= RNG_MIN)? ((range->begin)=val): (range->begin =RNG_MIN))

/**Macro***********************************************************************
  Synopsis     [Set the end of the range]
  Description  [Given a range and a value L, this sets the end value
   L of the range.]
  SideEffects  [There original ending value of the range is lost.]
  SeeAlso      [Rng_RangeSetBegin]
******************************************************************************/
#define Rng_RangeSetEnd(range,val)\
((val <= RNG_MAX)? ((range->end)=val): (range->end =RNG_MAX))

/**AutomaticStart*************************************************************/
/**AutomaticEnd***************************************************************/
#endif /* _RNG */
```

4.2 The Internal Header File `rngInt.h`

```
/**CHeaderFile*****************************************************************
  FileName    [rngInt.h]
  PackageName [rng]
  Synopsis    [This package deals with functions that are used to manipulate
  the range (Rng_Range_t) data structure.]

  Description [ The basic data structure in the range package
  is an integer range (range= {L,U}, where L is the lower limit of
  the range, and U is the upper limit of the range. The range {L,U}
  includes all integers x, L =< x =< U).]
  SeeAlso     [tbl.h rng.h]
  Author      [Gitanjali Swamy]
  Copyright   [Copyright (c) 1994-1996 The Regents of the Univ. of California.
  All rights reserved.]
  Revision    [$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $]
******************************************************************************/
#ifndef _RNGINT
#define _RNGINT
```

```
#include "rng.h"

/*---------------------------------------------------------------------------*/
/* Stucture declarations                                                     */
/*---------------------------------------------------------------------------*/

/**Struct**************************************************************
  Synopsis      [ This struct represents a range.]
  Description   [ This struct represents a range and has 2 fields;
                  the beginning and the end.]
  SeeAlso       [ Rng_RangeAlloc Rng_RangeFree ]
*******************************************************************/
struct RngRangeStruct {
    int begin; /* The beginning of the range */
    int end;   /* The end of the range */
};

/*---------------------------------------------------------------------------*/
/* Macro declarations                                                        */
/*---------------------------------------------------------------------------*/

/**Macro***********************************************************************
  Synopsis     [Get the maximum of two number]
  Description  [Given two integers u1 and u2, this will return the
  maximum of the two.]
  SideEffects  []
  SeeAlso      [RngRangesMin]
******************************************************************************/
#define RngRangesMax(u1, u2)\
(u1 > u2) ? u1 : u2

/**Macro***********************************************************************
  Synopsis     [Get the minimum of two number]
  Description  [Given two integers u1 and u2, this will return the
  minimum of the two.]
  SideEffects  []
  SeeAlso      [RngRangesMax]
******************************************************************************/
#define RngRangesMin(u1, u2)\
(u1 < u2) ? u1 : u2

/**AutomaticStart*************************************************************/
/**AutomaticEnd***************************************************************/
#endif /* _RNGINT */
```

4.3 rngUtil.c

```
/**CFile***********************************************************************
  FileName    [rngRangeUtil.c]
  PackageName [rng]
  Synopsis    [This file deals with utilities that are exported to
   handle the range Rng_Range_t data structure.]

  Description [ This file contains memory management utilities, and
  functions for oring together two ranges. The range data structure or
  Rng_Range_t has two fields; range->begin (the beginning of the
  range) and range->end (the end of the range). These are accessed using
  macros define in rng.h.]

  SeeAlso     [rng.h rngInt.h rngUtilInt.c]
  Author      [Gitanjali Swamy]
  Copyright   [Copyright (c) 1994-1996 The Regents of the Univ. of California.
  All rights reserved.]
******************************************************************************/
#include "rngInt.h"

static char rcsid[] = "$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $";
```

```
USE(rcsid);

/**AutomaticStart*************************************************************/
/**AutomaticEnd***************************************************************/

/*---------------------------------------------------------------------------*/
/* Variable declarations                                                     */
/*---------------------------------------------------------------------------*/
int Rng_NumRanges;

/*---------------------------------------------------------------------------*/
/* Definition of exported functions                                          */
/*---------------------------------------------------------------------------*/

/**Function********************************************************************
  Synopsis    [Allocates memory for a Rng_Range_t.]
  Description [This functions takes no inputs, and when called
  allocates space for and returns a new Rng_Range_t.]
  SideEffects []
  SeeAlso     [Rng_RangeFree]
******************************************************************************/
Rng_Range_t*
Rng_RangeAlloc()
{
  Rng_Range_t *range;

  range = ALLOC(Rng_Range_t,1);
  return range;
}

/**Function********************************************************************
  Synopsis    [Free memory associated with a Rng_Range_t.]
  Description [This function takes a Rng_Range_t* as input, and free
  the memory used by it.]
  SideEffects [Rng_Range_t is no longer valid.]
  SeeAlso     [Rng_RangeAlloc]
******************************************************************************/
void
Rng_RangeFree(Rng_Range_t *range)
{
  FREE(range);
}

/**Function********************************************************************
  Synopsis    [OR two ranges.]
  Description [Given two ranges, this function unions together the two,
  and returns the result. If the two ranges do not overlap it returns
  with an error flag.]
  SideEffects []
******************************************************************************/
Rng_Range_t *
Rng_RangeOrWithRange(Rng_Range_t* range1, Rng_Range_t* range2)
{
   Rng_Range_t* newrange;

  if ( !(RangeOverlap(range1,range2)) || !(RngRangeIsOK(range1)) ||
       !(RngRangeIsOK(range2)) ) {
     error_flag();
     return NIL(Rng_Range_t);
  } else {
    newrange = Rng_RangeAlloc();
    Rng_RangeSetBegin( newrange, RngRangesMin(Rng_RangeReadBegin(range1),
                       Rng_RangeReadBegin(range2)) );
    Rng_RangeSetEnd( newrange, RngRangesMax(Rng_RangeReadEnd(range1),
                       Rng_RangeReadEnd(range2)));
  }
 return newrange;
}
```

```
/*---------------------------------------------------------------------------*/
/* Definition of static functions                                            */
/*---------------------------------------------------------------------------*/

/**Function********************************************************************
  Synopsis    [Check if ranges overlap.]
  Description [Given two ranges of the type Rng_Range_t*, this
  function returns TRUE if they overlap, and FALSE if they do not.]
  SideEffects []
  SeeAlso     [Rng_RangeOrWithRange]
******************************************************************************/
static boolean
RangeOverlap(Rng_Range_t *range1, Rng_Range_t *range2)
{
 if ( (Rng_RangeReadBegin(range1) > (Rng_RangeReadEnd(range2) - 1)) ||
      (Rng_RangeReadBegin(range2) > (Rng_RangeReadEnd(range1) - 1)) ) {
   return 0;
 } else {
   return 1;
 }
}
```

4.4 rngIntUtil.c

```
/**CFile***********************************************************************
  FileName    [rngRangeIntUtil.c]
  PackageName [rng]
  Synopsis    [Internal package utilities in the range package.]
  Description [ This has just one internal function called RngRangeIsOk.]
  SeeAlso     [rng.h rngInt.h rngUtil.c]
  Author      [Gitanjali Swamy]

  Copyright   [Copyright (c) 1994-1996 The Regents of the Univ. of California.
  All rights reserved.]
******************************************************************************/
#include "rngInt.h"

static char rcsid[] = "$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $";
USE(rcsid);

/**AutomaticStart*************************************************************/
/**AutomaticEnd***************************************************************/

/*---------------------------------------------------------------------------*/
/* Definition of internal functions                                          */
/*---------------------------------------------------------------------------*/


/**Function********************************************************************
  Synopsis    [Check if a Rng_Range_t is well ordered.]
  Description [Given a Rng_Range_t , this function returns TRUE if it
  is a valid range, and FALSE if it is not.]
  SideEffects []
******************************************************************************/
boolean
RngRangeIsOK(Rng_Range_t *range)
{
  if (Rng_RangeReadBegin(range) <= Rng_RangeReadEnd(range)) {
    if (Rng_RangeReadBegin(range) => RNG_MIN) {
       if (Rng_RangeReadEnd(range) <= RNG_MAX) {
          return 1;
       }
    }
  }
  return 0;
}
```

Chapter 5

Maintenance Utilities

The VIS environment currently has two maintanance utilities; `extdoc`, which extracts documentation from source code, and `extproto`, which automatically inserts function declarations within header and source files.

VIS uses the Revision Control System, or RCS, for version control.

5.1 `extproto`

`Extproto` analyzes source code and inserts automatically-generated function declarations into the appropriate source and header files. It requires a package that follows the conventions in Chapters 2 and 3.

When run on a header file named *package*`.h` (the external header file), it looks for externally-visible functions (i.e., those with the prefix *Package_*) and places them between the `/**AutomaticStart***` and `/**AutomaticEnd***` comments within the header file described in Chapter 2.

When run on a header file named *package*`Int.h` (the internal header file), it looks for internally-visible functions (i.e., those with the prefix *Package*) and places them between the automatic comments within the file.

When run on a C source file, it looks for `static` functions and places them between the `/**AutomaticStart**` and `/**AutomaticEnd` comments.

For example, if `extproto` is run on the `rng` example described in the previous section, then it will add lines to the files in the package as shown.

- To extract the `static` functions in `rngUtil.c`,

  ```
  % extproto rngUtil.c
  processing rngUtil.c
  %
  ```

  which adds the following lines to `rngUtil.c`:

  ```
  /**AutomaticStart*************************************************************/

  /*---------------------------------------------------------------------------*/
  /* Static function prototypes                                                */
  /*---------------------------------------------------------------------------*/

  static boolean RangeOverlap(Rng_Range_t *range1, Rng_Range_t *range2);

  /**AutomaticEnd***************************************************************/
  ```

- To extract the internal and external function prototypes into the appropriate headers,

  ```
  % extproto *.h
  processing rng.h
  processing rngIntUtil.c
  processing rngUtil.c
  Writing rng.h, the external header
  processing rngInt.h
  processing rngIntUtil.c
  processing rngUtil.c
  Writing rngInt.h, the internal header
  %
  ```

  which adds the following lines to `rng.h`, the external header

  ```
  /**AutomaticStart*************************************************************/

  /*---------------------------------------------------------------------------*/
  /* Function prototypes                                                       */
  /*---------------------------------------------------------------------------*/

  EXTERN Rng_Range_t* Rng_RangeAlloc();
  EXTERN void Rng_RangeFree(Rng_Range_t *range);
  EXTERN Rng_Range_t* Rng_RangeOrWithRange(Rng_Range_t* range1,
    Rng_Range_t* range2);

  /**AutomaticEnd***************************************************************/
  ```

  and the the following lines to `rngInt.h`, the internal header

  ```
  /**AutomaticStart*************************************************************/

  /*---------------------------------------------------------------------------*/
  /* Function prototypes                                                       */
  /*---------------------------------------------------------------------------*/

  EXTERN boolean RngRangeIsOK(Rng_Range_t *range);

  /**AutomaticEnd***************************************************************/
  ```

5.2 `extdoc`

`Extdoc` extracts documentation from packages that follow the conventions in Chapters 2 and 3. It can write both simple text files, or HTML for the World Wide Web.

For example, running `extdoc` on the `rng` package of Chapter 4 in text-producing mode,

```
% extdoc --text rng
processing rngRangeIntUtil.c
processing rngRangeUtil.c
processing rngInt.h
processing rng.h
writing rngDoc.txt
%
```

creates a file `rngDoc.txt` that contains

```
The rng package

This package deals with functions that are used to manipulate   the range
(Rng_Range_t) data structure.
```

```
Gitanjali Swamy

*******************************************************************

Rng_RangeAlloc()              Allocates memory for a Rng_Range_t.

Rng_RangeFree()               Free memory associated with a Rng_Range_t.

Rng_RangeOrWithRange()        OR two ranges.

Rng_RangeReadBegin()          Return the beginning of the range.

Rng_RangeReadEnd()            return the end of the range.

Rng_RangeSetBegin()           Set the beginning of the range.

Rng_RangeSetEnd()             Set the end of the range

*******************************************************************

The basic data structure in the range package is an   integer range. The
range is designated as {L,U}, where L is the lower limit of    the range, and
U is the upper limit of the range, and     includes all integers x, L =< x =<
U).

Rng_Range_t*
Rng_RangeAlloc(

)
  This functions takes no inputs, and when called allocates space for and
  returns a new Rng_Range_t.


void
Rng_RangeFree(
  Rng_Range_t *     range
)
  This function takes a Rng_Range_t* as input, and free the memory used by it.

  Side Effects: Rng_Range_t is no longer valid.

Rng_Range_t *
Rng_RangeOrWithRange(
  Rng_Range_t*      range1,
  Rng_Range_t*      range2
)
  Given two ranges, this function OR's together the two, and returns the
  result. If the two ranges do not overlap it returns with an error flag.



Rng_RangeReadBegin(
                    range
)
  Given a range {L,U}, this returns the beginning value L of the range.



Rng_RangeReadEnd(
                    range
)
  Given a range {L,U}, this returns the ending value U of the range.



Rng_RangeSetBegin(
                    range,
                    val
)
```

```
  Given a range and a value L, this sets the beginning value L of the range.

  Side Effects: There original beginning value of the range is lost.


Rng_RangeSetEnd(
                    range,
                    val
)
  Given a range and a value L, this sets the end value L of the range.

  Side Effects: There original ending value of the range is lost.
```

5.3 RCS

RCS is a file-based revision control system used within VIS.

To use RCS within VIS,

1. Create a directory `~/vis/common/src`.
2. Define the environment variables `VIS` and `VIS_USER_DIR`:

   ```
   % setenv VIS /projects/vis/vis
   % setenv VIS_USER_DIR ~/vis/common/src
   ```

3. To put a new package `pkg` residing in `$VIS/common/src/pkg` under RCS control,

   ```
   % mkdir $VIS/common/src/pkg/RCS
   % chmod 775 !$
   % echo "VIS: Initial version" > /tmp/rcsText
   % cd $VIS/common/src/pkg
   % gmake RCSFLAGS='-t/tmp/rcsText' rcs_ci
   ```

The files in the packages in `$VIS/common/src` are not writable. All changes must be made through RCS, using the working area `$VIS_USER_DIR`.

The `Makefile` in each package has 5 RCS commands defined. These commands operate on all the files of the package under RCS control, namely `RCSFILES = $(SRC) $(HDR) $(MISC) $(LFILE) $(YFILE)` as defined in the `Makefile`.

To use the commands on package `pkg`, first create directory `pkg` in `$VIS_USER_DIR`. Then, you can execute package level RCS commands from either the `Makefile` at `$VIS/common/src/pkg`, or the `Makefile` at `$VIS_USER_DIR/pkg` (if you already have a `Makefile` there). The commands are as follows:

- `gmake rcs_co`

  Check out a copy of `RCSFILES` and put them at `$VIS_USER_DIR/pkg`. This operation will fail if another user has these files checked out. In this case, negotiate with this user to determine when you can have access. To keep things simple, we don't want to have multiple branches of development.

- `gmake rcs_ci`

  Check in the `RCSFILES` in `$VIS_USER_DIR/pkg` into `$VIS/common/src/pkg/RCS`. This leaves a copy of the new files at `$VIS/common/src/pkg`. For any file that has changed, you will be prompted to supply a log message (terminated by `ctrl-D` or a period on a line by itself). For files that have not changed, the check-in does nothing. This operation causes the protection of `RCSFILES` in `$VIS_USER_DIR/pkg` to be changed to read only. You can choose to remove (`rm`) these files, or leave them there. If you `make rcs_co` again, these files will be overwritten and the protection will be changed to allow writing.

- `gmake rcs_diff`

  For each `RCSFILE`, performs a diff between the corresponding files in `$VIS_USER_DIR/pkg` and `$VIS/common/src/pkg/RCS`. This is useful to remind yourself what changes (if any) you have made.

- `gmake rcs_ident`

  For each `RCSFILE` in `$VIS/common/src/pkg`, searches the file for RCS identifiers, like `$Id`. This is useful to see who most recently modified a file, and when this was done.

- `gmake rcs_status`

  For each `RCSFILE` in `$VIS/common/src/pkg` currently checked out, this tells you who has the file checked out. This is useful to determine who is modifying which files.

The first two commands do all the work. The last three are only for informational purposes; you can't do any harm by using them.

More experienced RCS users can specify options to the RCS commands by setting the `RCSFLAGS` in the `gmake` invocation (e.g., `gmake RCSFLAGS='-b -i' rcs_diff`).

RCS commands can be performed from the `Makefile` at `$VIS/common/src`:

- `gmake rcs_ident`: Run `make rcs_ident` on each package.

- `gmake rcs_status`: Run `make rcs_status` on each package.

For files not part of code packages but under RCS control (e.g., `src/Makefile`, `ext/*`), you must use RCS commands on individual files; there is no `Makefile` support for them.

Chapter 6

Compiling VIS and Creating a New Package

6.1 Compiling VIS

Standard `Makefiles` are provided with each package within VIS. To compile

1. a personal version of VIS using your modifed sources:

   To build a VIS executable that includes a package in your private area (e.g., one you checked out and modified), run `gmake CC=gcc vis-g` there. This will create an executable called `vis-g` by compiling your source files and linking them with the public `libvis-g.a`.

2. a new public version of VIS in the central area:

   Check in any modified packages and rebuild the executable and libraries by running `gmake CC=gcc vis-g` in the `/projects/vis/vis-devel/`*machine*`/src` directory. (*machine* is `mips`, `alpha`, etc.) The executable will be placed in `/projects/vis/vis-devel/`*machine*`/bin`; the libraries in `/projects/vis/vis-devel/`*machine*`/lib`.

For portability and uniformity, please use the suggested compiler `gcc`.

6.2 Adding Code to VIS

To add code to the VIS system (that may or may not become part of the public system), use the `tst` package. This dummy package is known to the VIS system, yet is easily adapted to your purposes. If later your package is included in the public system, it can easily be adapted to become permanent.

The VIS system knows about two functions in the `tst` package:

- `Tst_Init`. This is called when the VIS system starts, and should register your new commands by calling `Cmd_CommandAdd`.

- `Tst_End`. This is called when the VIS system terminates, and should free any global memory allocated by your package.

For example, to add a command _testRng to VIS, use the following function

```
/**Function********************************************************************
  Synopsis    [Implements the _test_rng command.]
  SideEffects []
******************************************************************************/
static int
CommandTestRng(
  Hrc_Manager_t ** hmgr,
  int  argc,
  char ** argv)
{
  int c;
  int verbose = 0;              /* default value */

  /*
   * Parse command line options.
   */

  util_getopt_reset();
  while ((c = util_getopt(argc, argv, "v")) != EOF) {
    switch(c) {
        case 'v':
          verbose = 1;
          break;
        default:
          goto usage;
    }
  }


  if (verbose) {
    (void) fprintf(vis_stdout, "The _test_rng command does nothing\n");
  }

  /*
   * Normal exit
   */

  return 0;

usage:
  (void) fprintf(vis_stderr, "usage: _test_rng [-v]\n");
  (void) fprintf(vis_stderr, "    -v\t\tverbose\n");

  /*
   * Error exit
   */

  return 1;
}
```

and change Tst_Init and Tst_End (in tst.c) as follows

```
/**CFile***********************************************************************
  FileName    [tst.c]
  PackageName [tst]
  Synopsis    [Test package initialization, ending, and the command _test.]
  Author      [Originated from SIS.]
  Copyright   [Copyright (c) 1994-1996 The Regents of the Univ. of California.
  All rights reserved.
******************************************************************************/

#include "tstInt.h"

static char rcsid[] = "$Id: vis_eng.tex,v 1.25 1996/05/11 00:05:49 shiple Exp $";
USE(rcsid);
```

```
/*---------------------------------------------------------------------------*/
/* Definition of exported functions                                          */
/*---------------------------------------------------------------------------*/

/**Function********************************************************************
  Synopsis    [Initializes the test package.]
  SideEffects []
  SeeAlso     [Tst_End]
******************************************************************************/
void
Tst_Init()
{

  /*
   * Add a command to the global command table.  By using the leading
   * underscore, the command will be listed under "help -a" but not "help".
   */

  Cmd_CommandAdd("_test_rng", CommandTestRng, /* doesn't changes_network */ 0);
}

/**Function********************************************************************
  Synopsis    [Ends the test package.]
  SideEffects []
  SeeAlso     [Tst_Init]
******************************************************************************/
void
Tst_End()
{
  /*
   * For example, free any global memory (if any) which the test package is
   * responsible for.
   */
}
```

6.3 Adding a New Package to VIS

To add a new package to VIS,

1. Make sure your code compiles and runs within VIS, and that it conforms to the conventions in this document.

2. Create a directory in `/projects/vis/vis-devel/common/src` named `package` (e.g., `rng`).

3. Add your package to the list of packages (`PKGS =`) in `/projects/vis/vis-devel/common/src/Makefile`.

4. Make a symbolic link in the `/projects/vis/vis-devel/common/include` directory that points to your package's external header file (e.g., `ln -s ../src/rng/rng.h`).

5. Add a `#include` in `/projects/vis/vis-devel/common/src/vm/vm.h` for your package's external header file (e.g., `#include "rng.h"`).

6. Add calls to your package's `Init` and `End` functions in `/projects/vis/vis-devel/common/src/vmVinit.c`.

7. Make symbolic links in `/projects/vis/vis-devel/`*machine*/src to the source files in your package. *Copy* the `Makefile` to these directories, since it will be regenerated to reflect architecture-specific dependencies.

VIS Engineering Manual
Quick Reference

Package Filenames

Name	Contains
pkg.h	Externally-visible functions, etc. Includes system headers only
pkgInt.h	Functions, etc., internal to the package. Includes pkg.h, pkgPort.h if present.
pkgPort.h	#defines, etc., that hide system differences
pkgUtil.c	E.g., utility functions. #include "pkgInt.h"
pkgIntUtil.c	E.g., internal utility functions. #include "pkgInt.h"

Identifier Names

	External	Internal	Local
Functions, Macros	Pkg_SubjectVerbModifier	PkgSubjectVerbModifier	SubjectVerbModifier
Variables	pkg_numThings	pkgNumThings	numThings
Structures	Pkg_Thing_t	PkgThing_t	Thing_t
Enumerated types	Pkg_ThingList_c	PkgThingList_c	ThingList_c
Constants	PKG_SOME_THING	PKGSOME_THING	SOME_THING

RCS-VIS Commands

Name	Function
rcs_co	Check out a copy of RCSFILES into $VIS_USER_DIR/pkg
rcs_ci	Check in RCSFILES in $VIS_USER_DIR/pkg into $VIS/common/src/pkg/RCS
rcs_diff	Find difference between RCSFILES in $VIS/common/src/pkg/RCS and $VIS_USER_DIR/pkg
rcs_ident	Prints out RCS identifiers (like $Id) of RCSFILES in $VIS/common/src/pkg/RCS
rcs_status	Print out last user who modified the RCSFILES in $VIS/common/src/pkg