The GENUS User Manual and C Programming Library

(Document *Revision* : 4.0, UCI-CADLAB)

Pradip K. Jha and Nikil D. Dutt

Technical Report# 93-32
(Revision 4.0)
August 23, 1994

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
Phone: (714) 856-8059
FAX: (714) 856-4056
Email: pradip@ics.uci.edu

Abstract

This document first gives a brief overview of the GENUS library of Generic Component Generators and describes the installation procedure for the library. It then presents the C-programming library interface and illustrates its use with some examples. The appendix has a complete list of the GENUS generators and some sample models.

Contents

List of Figures

List of Tables

1 Introduction

This document first gives a brief overview of the GENUS library of Generic Component Generators and describes the installation procedure for the library. It then presents the programming library interface and illustrates its use with some examples. The appendix has a complete list of the GENUS generators and some sample models.

The GENUS library provides a standardized set of RT components that have well-defined semantics along with models to facilitate simulation and synthesis. GENUS consists of a set of parametrized RT component generators. These generators are defined using RT-level functionality and are grouped into classes based on functional similarity. Each generator is associated with a set of parameters; a component instance is generated by specifying values to these parameters. GENUS also allows users to define and store custom components alongwith their simulation and synthesis models.

This implementation of GENUS consists of a self-contained library of routines for creating and retrieving information from the parameterizable component generators (also called "compilers"), components, and component instances ("carbon-copies" of a specific component).

This version (4.0) covers the complete set of components discussed in [DuJh92], alongwith model generators for simulation and synthesis. It also provides generic gate and flip-flop counts used in the synthesis model. A mechanism to integrate the performance metrics (e.g., area, delay, etc.) for GENUS components has been added. These metrics can either be incorporated from a technology library or can be derived from on-line estimators that generate estimates given a component's parameters [JhDu93]. GENUS 4.0 also allows shape functions (length-width-delay tuples) to be stored with every component.

2 GENUS System Overview

The organization of the GENUS library is illustrated in Figure 1. The core of the library is the set of component generators. These generators contain information regarding the type, parameters, ports and functionality of the components under them. The component library is supplemented by query routines, performance metric routines, behavioral model, synthesis model and gate-count generators. An application C-program such as High-level Synthesis tool (e.g., [LiGa88][RaGa91])interacts with GENUS by assigning a component's parameter values. The GENUS library uses these parameters to initially create a component and delete the component when necessary. GENUS also provides query routines to retrieve an instantiated component's attributes (e.g., port information, metrics) as well as to generate behavioral VHDL models for simulation and logic equation models for synthesis. This section briefly describes the GENUS structure; details of the GENUS interface is provided later in this report. Appendix D lists the complete set of component generators. For detailed information regarding GENUS, see [DuJh94].

2.1 Hierarchy

GENUS is organized into 4 levels of hierarchy, where each level inherits attributes from its parent level. Figure 2 shows a sample GENUS snapshot, where instances I1 through I4 are children of the class of 4-bit ALU components. The ALU components are generated from the class of ALU

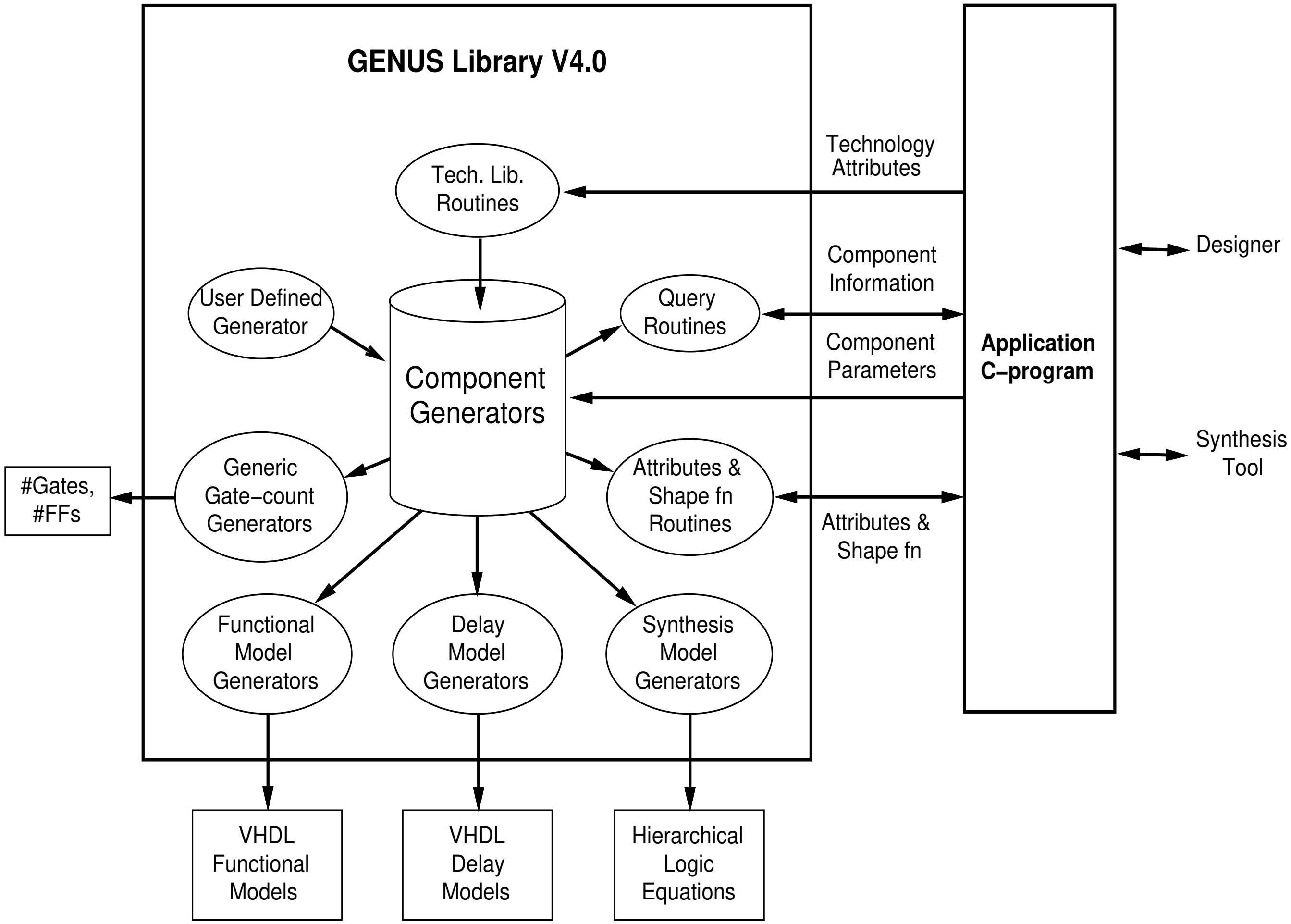

Figure 1: Organization of GENUS System

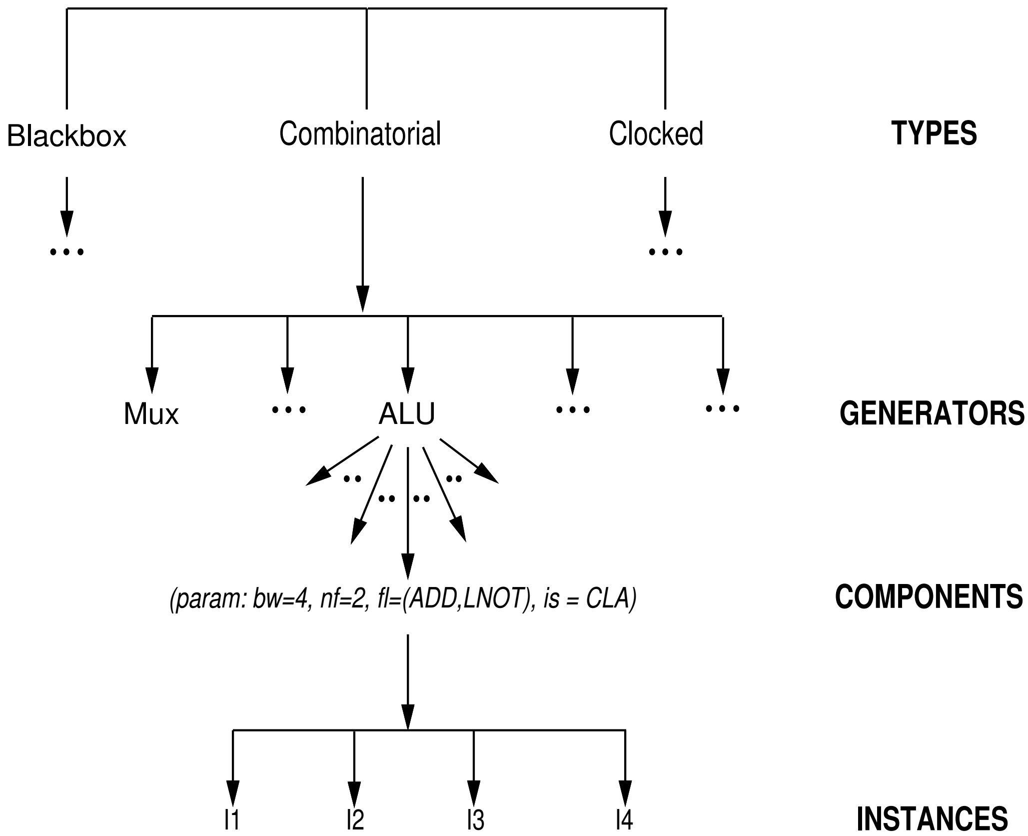

Figure 2: Hierarchy in GENUS

generators by assigning values to ALU parameters (*bit-width = 4, num-functions = 2, function-list = (ADD, LNOT), implementation-style = CARRY_LOOKAHEAD).* Finally, the ALU generator class belongs to the *combinatorial* type class. Each of these classes is described below:

Types The type class describes the abstract functionality of elements in GENUS. Currently, GENUS supports three types:*Combinatorial, Clocked and Blackbox.* The *Blackbox* type refers to the user-defined components.

Generators A generator class is used to generate a family of similar components and instances. Briefly, a generator descriptor maintains a list of all the possible parameters, definitions for each operation performed by a generated component. Appendix D describes the format of a generator description and also lists the complete set of generators.

Components A component is generated by passing a list of parameters to the parent generator. For instance, the ALU component in Figure 2 is generated by specifying these parameters: (*bit-width = 4, num-functions = 2, function-list = (ADD, LNOT), implementation-style = CARRY_LOOKAHEAD*). All possible parameters for a particular generator need not be specified; optional parameters are assigned default values.

Instances Instances are "carbon-copies" of a generated component, with different names and values to performance metrics (ATTRIBUTES) such as area, delay, etc. These GENUS elements are the ones actually used for connectivity in the structural design. Since an instance inherits all of its parameters from the parent component, only the connectivity of the instance is stored in its representation.

2.2 Generator Parameters

As mentioned before, each generator in GENUS is associated with a set of parameters that are used to instantiate components. The domain of parameters characterize the set of components encompassed by a specific generator. For example, an ALU generator is characterized by the following parameters: *(bit-width, num-functions, functions-list, implementation-style)*, whereas a specific ALU instance is generated by assigning values to these parameters. This grouping of similar components into classes of generators makes the task of library management simpler and more efficient since the resulting number of the generators (approximately 50) is much less as compared to the virtually infinite number of component instantiations. Since the concept of generator parameters is crucial to interaction with GENUS library, we present further details of parameters in Section 4.1.

2.3 Global Features and Default Values

The GENUS system, as shown in Figure 1, consists of various modules or *features* (shown in the oval boxes), namely functional model generators, delay model generators, synthesis model generators, technology library routines and gate-count generators. GENUS provides a global routine to switch on or off each of these *features*. This routine customizes the GENUS system (thus making it more efficient) to a particular scenario that requires only a subset of these features. Also, the default value for the *data-type* (BIT/MVL4) to be used for simulation models can be selected by the user. These routines to select the global features and data-type are discussed in Section 11.

2.4 Snapshots of a Sample Interaction Scenario

In a typical session with GENUS, a user should be able to create a component, query information, read technology attributes from library and generate models and generic gate-count. This is accomplished by invoking C-routines form the GENUS programming library. The following steps illustrate a sample scenario:

1. Initialize GENUS to set up the data structures.

```
GENUS_initialize();
```

2. Switch on/off a feature in GENUS.

```
ret_value = GENUS_setFeature(GF_FUNCTIONAL_MODEL, False);
```

3. Create a component. This is done by assigning values to the parameters.

```
function_list[0] = GC_ADD;
function_list[1] = GC_LNOT;
i  = 0;
GENUS_setParameter(list[i], GC_COMPILER_NAME, "ALU"); i++;
GENUS_setParameter(list[i], GC_INPUT_WIDTH, 4); i++;
GENUS_setParameter(list[i], GC_NUM_FUNCTIONS, 2);i++;
GENUS_setParameter(list[i], GC_FUNCTION_LIST, function_list); i++;
GENUS_setParameter(list[i], GC_STYLE, GC_CARRY_LOOKAHEAD); i++;
GENUS_setParameter(list[i], GC_END_LIST, 0); i++;
alu_component = GENUS_createComponent(GC_PARAMETER_LIST, list);
```

4. Instantiate components alongwith technology attributes from a library. This is accomplished by specifying the environment variable GENUS_LIBRARY_DIR.

5. Create an instance of a component.

```
i  = 0;
GENUS_setParameter(list[i], GC_COMPONENT_PTR, alu_component); i++;
GENUS_setParameter(list[i], GC_INSTANCE_NAME, "ALU1"); i++;
GENUS_setParameter(list[i], GC_ATTRIBUTE_PTR, at); i++;
GENUS_setParameter(list[i], GC_END_LIST, 0); i++;
alu_instance = GENUS_createInstance(GC_PARAMETER_LIST, list);
```

6. Query information such as the number of input ports for an instantiated component.

```
num_inputs =  GCQ_InstInt(alu_instance, GC_NUM_INPUTS);
```

7. Retrieve performance metrics for a component(e.g., for a GENUS component instance, find the best area implementation).

```
GENUS_getBestAreaAttribute(GC_COMPONENT_PTR, alu_component);
```

8. Generate functional model for a component. This is generated automatically whenever a component is instantiated.

9. Generate delay model for a component. This is generated automatically whenever an instance is created.

10. Generate synthesis model (hierarchical logic equations) for a component. This is generated automatically whenever a component is instantiated.

11. Generate generic gate and flip-flop counts. This is generated automatically whenever a component is instantiated.

12. Delete a component.

```
GENUS_delete(GC_COMPONENT_PTR, alu_component);
```

The above snapshots were extracted from the detailed tutorial examples present in Appendix A.

In the rest of this report, we discuss the GENUS user interface including the details of how to perform the above mentioned tasks. Section 3 discusses installation procedures. Section 4 explains the GENUS parameters and the routines for creating and deleting components and instances. Section 5 discusses the query routines to retrieve information for instantiated components; Section 7 describes the routines to incorporate performance metrics. Routines to handle complex parameter types are described in Section 8. Section 9 discusses the VHDL functional and delay models and 10 discusses synthesis models and generic gate-count generation. Finally, Section 11 lists some miscellaneous routines.

3 Installing and using GENUS

This version of GENUS has been tested on SUN-4 platforms with SUN OS 4.1.1. GENUS requires around one Megabytes of disk space.

In order to load the software on your system complete the following steps:

1. Insert the GENUS Distribution Tape into the tape drive of SUN-4 machine.

2. Select a directory for installing the GENUS software. We shall refer to the pathname of this directory by *instdir*. Make sure that *instdir* has the appropriate write permissions. Move to the installation directory.

```
% cd instdir
```

3. The GENUS library is loaded into a subdirectory named *bin*. The include files are copied into the subdirectory *include*. Make sure that the *instdir* has the proper write permission and that it does not contain the above mentioned subdirectories, otherwise it will be overwritten.

4. Load the GENUS software into the computer's file system by typing,

```
% tar xvf /dev/rst8
```

5. Remove the tape from the drive. You can check to see if the installation was successful by listing the files that were loaded.

```
% ls -l
.......... README
.......... bin/
.......... include/
.......... vhdl/
```

3.1 Compiling and Linking

A user interfaces with GENUS through a set of functions and procedures, as described before. In order to use these routines, the GENUS library has to be linked while compiling the user's C-source code. The GENUS library and function prototypes are loaded in *instdir/bin* and *instdir/include* respectively. The file `GENUS.h` of type definitions should be included in each application source file.

Thus, an application source file using the GENUS generic component generators would be compiled with:

```
cc -Iinstdir/include -Linstdir/bin -lGENUS [filename.c]
```

Before using GENUS, the user should set these environment variables: `GENUS_VHDL_DIR` and `GENUS_LOGIC_DIR` to the directory to which the user wants VHDL behavioral and the logic synthesis models to be directed to. Otherwise, GENUS will assume that the directory containing the source code has subdirectories named *vhdl_models* and *logic_equations*. The environment variables can be set by the following unix commands:

```
setenv GENUS_VHDL_DIR vhdl_dir_name
setenv GENUS_LOGIC_DIR logic_dir_name
```

These directories maintain the VHDL simulation models and the synthesis logic equations for the accessed components. Since these models are generated automatically for each new component, the user should periodically clean up these directories by deleting the files that are not required.

4 Basic GENUS Interface

4.1 Parameters and Types

The user interacts with the GENUS programming routines by specifying a component's parameter name, parameter values, and parameter types (*type* has the same meaning here and throughout as that used by the C language). For example, the first parameter name/value pair passed to the routine **GENUS_createComponent** to create a generic multiplexor is

`GC_COMPILER_NAME, "MUX"`

In this example, `GC_COMPILER_NAME` is the parameter name and `"MUX"` is the parameter value. The parameter type is `(char *)`.

4.1.1 Parameter Names

Parameter names accompanied by parameter values are used to describe everything about GENUS component generators, components, and instances. They are defined in C as enumerated constants and thus are entered exactly as they appear. Parameter names symbols are of the type definition `GENUS_parameterName`.

Parameter names always precede parameter values.

A parameter name does not necessarily have to have a fixed parameter value type. For example, although `GC_NUM_INPUTS` will always describe an integer value, `GC_FUNCTION_LIST` may describe either a list of enumerated constants, e.g. `GC_AND`, `GC_OR`, and `GC_NOT`, for an ALU, or a list of character strings, e.g. "I0", "I1", and "I2", as would be the case for a 3 input MUX.

Appendix C.1 lists all of the parameter names and value types.

4.1.2 Parameter Types and Values

A parameter type in GENUS may correspond directly to a simple C data type, describe an enumerated constant list, or point to a predefined type definition structure (the latter are described in Section 8, since they require additional query routines). For example, `GC_Int` is a GENUS type symbol corresponding to the C type `int`, while `GC_CharPtrPtr` corresponds to the more complex `GENUS_charPtrPtrType *` type definition.

Since procedures may require (or return) type names it is necessary to have symbols which refer to the specific data types, since C doesn't allow type definitions to be passed as arguments. Table 1 lists all of the GENUS type symbols and their corresponding C type definitions. The type symbols are of the type definition `GENUS_parameterType`.

The following list describes some of the simple types that appear in the second column of table 1.

- `GENUS_miscParameterValue`

  This type is an enumerated list of constants that corresponds to the value for parameters that require a single item from a set of possible items. For example, the ALU generator requires the `GC_STYLE` parameter, which can have one of the `GENUS_miscParameterValue` values `GC_RIPPLE_CARRY`, `GC_CARRY_LOOKAHEAD` or `GC_INTEGRATED`. Appendix C.6 describes the complete `GENUS_miscParameterValue` list.

- `GENUS_class`

  This enumerated constant type contains the values permitted to the `GC_CLASS` parameter. Its current values are two: `GC_Combinatorial` and `GC_Clocked`.

- `GENUS_operation`

  This enumerated constant type contains all of the data operations available. Appendix C.7 describes the list.

- `Bool`

  This type defines zero as `False` and non-zero as `True`. (It is actually type `char` so care should be taken when mixing this type with conventional integer boolean arithmetic.)

GENUS Type Symbol	**C Type Name**
GC_TypeUndef	(*undefined type error*)
GC_Int	int
GC_Misc	GENUS_miscParameterValue
GC_MiscList	GENUS_miscListType * †
GC_EMiscList	GENUS_miscParameterValue *
GC_Class	GENUS_class
GC_Operation	GENUS_operation
GC_OpClasses	GENUS_opClassesListType * †
GC_IntPtr	GENUS_intPtrType * †
GC_EIntPtr	int *
GC_IntPtrPtr	GENUS_intPtrPtrType * ††
GC_EIntPtrPtr	int ** ††
GC_Boolean	Bool
GC_CharPtr	char *
GC_CharPtrPtr	GENUS_charPtrPtrType * †
GC_ECharPtrPtr	char **
GC_CharPtrPtrPtr	GENUS_charPtrPtrPtrType * †
GC_ECharPtrPtrPtr	char ***
GC_ParameterValue	GENUS_parameterValue * ††
GC_ParameterList	GENUS_parameterListType * ††
GC_Proc	(**Boolean)() ††
GC_Generic	caddr_t (*generic pointer*)
GC_InstanceList	GENUS_instanceListType * †
GC_InstancePtr	GENUS_instance *
GC_ComponentKey	GENUS_componentKey †
GC_ComponentPtr	GENUS_component *
GC_OperationList	GENUS_operationListType * †
GC_EOperationList	GENUS_operation * †
GC_AttributeList	GENUS_attributeListType * †‡
GC_AttributePtr	GENUS_attribute * ‡
GC_EAttributeList	GENUS_attribute * †
GC_ShapePtr	GA_SHAPE * †

†Described in Section 8.
††Currently not for external use.
‡Described in Section 7.

Table 1: GENUS Type Symbols

- caddr_t

  For those familiar with ANSI C, this type corresponds semantically to void *. It is meant to convey the idea of generic pointer assignments (although, strictly speaking, it defeats the purpose of type checking).

- GENUS_componentKey

  This type exists solely to represent the hash string identifying a GENUS component. It is actually type char *.

- GENUS_attribute

  This type is a record to handle the performance metrics such as area, delay, etc.

- GA_SHAPE

  This type is a pointer to record that stores the length-width-delay tuple for a component's shape function.

4.1.3 Internal and External Parameter Types

Some type symbols in Table 1 have the letter E before the type description. These refer to external parameter types and only appear before complex types that require more than one variable to completely represent the corresponding value. Section 8 describes the complex parameter types in more detail. The remaining type symbols are either internal or function equivalently as both internal and external types.

External parameter types define complex data structures that are passed in to the creating routines. Once a component or instance has been created the query routines will then only return the corresponding internal type.

The reason for handling complex parameter values this way is largely due to non-tabular string lists. Consider how the following input list (from the SHIFTER generator) could be understood by an application program, were it not for complex parameter value types:

```
"I0[0]", "I0[1]", "I0[2]",
"ILIN[0]",
"IRIN[0]"
```

Clearly, three values are needed – one for each list. While the length of the first list is equivalent to a query of GC_INPUT_WIDTH, and the total number of lists is equal to GC_NUM_INPUTS, there is no simple, elegant way to determine the length of the last two lists.

The GC_EXTERNAL_PARAMETERS field in each generator (Appendix D) describes the relationship between parameter values passed to the creating routines and those returned from the query routines. For example, the entry for LU contains the line

GC_FUNCTION_LIST, ..., GC_EOperationList, GC_OperationList,

This indicates that the LU generator expects type GENUS_operation * to be passed into the creating routine and will return type GENUS_operationListType * from a query routine passed GC_FUNCTION_LIST.

4.2 Initializing the GENUS Component Library

Before the GENUS library routines can be used the generators must be initialized and various internal data structures set up. The following routine must be called once for each application before any other GENUS procedure.

Bool **GENUS_initialize**()

The return value indicates whether or not initialization was successful. A return value of `False` (0) may indicate that a generator was not written correctly.

The following routine can be used to determine whether or not GENUS has been initialized.

Bool **GENUS_initialized**()

4.3 Creating Generic Components

GENUS components are created by passing a list of parameter name/value pairs to the procedure **GENUS_createComponent**. (Note: components may also be created by **GENUS_createInstance**) An internal hashing scheme ensures that no component is created twice. If the list of parameters passed to **GENUS_createComponent** is identical to that of a component that has already been created the existing component will be returned. No error or warning message is given in this case.

There are two ways to call GENUS_createComponent. The first uses a variable argument list which must be terminated with `GC_END_LIST`.

GENUS_component * **GENUS_createComponent** (`GC_COMPILER_NAME`, *name*, *parameter_name*, *parameter_value*, ..., ..., `GC_END_LIST`)
char * *name*;
GENUS_parameterName *parameter_name*;
[value type] *parameter_value*;

The argument *name* refers to a GENUS generator name. Each *parameter_name* /*parameter_value* pair must correspond to a parameter name and type listed in the `GC_EXTERNAL_PARAMETERS` field of the requested generator (see Appendix D). Unrecognized parameters are ignored, but missing parameters will result in an error message and NULL returned.

Since a variable argument list cannot be constructed at program runtime the second call to **GENUS_createComponent** allows a previously defined list of parameter name/value pairs to be passed as a single argument.

GENUS_component * **GENUS_createComponent** (`GC_PARAMETERS`,*parameter_list*)
GENUS_externalParameter *parameter_list[]*;

The first parameter in *parameter_list* must be `GC_COMPILER_NAME` and the corresponding value the name of the compiler. Also, similarly to the first call, the last parameter in *parameter_list* must have name equal to `GC_END_LIST` (value is irrelevant).

The `GENUS_externalParameter` type is a structure containing two fields,

```
GENUS_parameterName name;
caddr_t value;
```

An individual parameter can be initialized with the **GENUS_setParameter** macro which has the following format:

void **GENUS_setParameter** (*parameter*,*name*,*value*)
  GENUS_externalParameter *parameter*;
  GENUS_parameterName *name*;
  [value type] *value*;

Thus, a list of parameters can be built up and passed to **GENUS_createComponent**. The following example illustrates this procedure.

```
create_alu()
{
   GENUS_component              *alu_component;
   GENUS_externalParameter      list[6];
   GENUS_operation              function_list[2];
   int                          i;

   function_list[0] = GC_ADD;
   function_list[1] = GC_LNOT;
   i=0;
   GENUS_setParameter(list[i], GC_COMPILER_NAME, "ALU"); i++;
   GENUS_setParameter(list[i], GC_INPUT_WIDTH, 4); i++;
   GENUS_setParameter(list[i], GC_NUM_FUNCTIONS, 2); i++;
   GENUS_setParameter(list[i], GC_FUNCTION_LIST, function_list);i++;
   GENUS_setParameter(list[i], GC_STYLE, GC_CARRY_LOOKAHEAD);   i++;
   GENUS_setParameter(list[i], GC_END_LIST, 0); i++;

   alu_component = GENUS_createComponent(GC_PARAMETER_LIST, list);
}
```

Note that the value of variable `i` is not and should not be incremented within the macro, e.g. `GENUS_setParameter(list[i++] ...)`. This is because the C preprocessor expands the macro into several references to its first argument which would result in `i` getting incremented twice. (For those familiar with the X toolkit – compare XtSetArg)

4.4 Creating Instances

GENUS instances are “carbon-copies” of a generated component, with differing names and performance metrics such as area, delay, etc. An instance inherits all its attributes from the parent component. A call to **GENUS_createInstance** creates an instance and adds it the component’s list of instances.

There are several instance-specific parameter names discussed later in this section.

There are several ways to call **GENUS_createInstance**, although only the first and last are strictly needed. The first call takes a pointer to a GENUS component (`GENUS_component *`) as its first parameter and a list of instance-specific parameter name/value pairs, and returns an instance pointer (`GENUS_instance *`). The variable argument list must be terminated with `GC_END_LIST`. The instance-specific parameters are listed in Section 4.4.1.

GENUS_instance * **GENUS_createInstance** (`GC_COMPONENT_PTR`, *comp*, *parameter_name*, *parameter_value*, ..., ..., `GC_END_LIST`)

GENUS_component *	*comp*;
GENUS_parameterName	*parameter_name*;
[type]	*parameter_value*;

The argument *comp* refers to a previously created GENUS component. Each *parameter_name/parameter_value* pair must correspond to an *instance-specific* parameter name and type, respectively.

The second call is similar to the first but takes `GC_COMPONENT_KEY` as the first argument and a value of type `GENUS_componentKey` as the second.

The third call is identical to the first way to call **GENUS_createComponent**, described in Section 4.3, although it may contain additional instance-specific parameters. (It exists primarily as an accelerator.)

GENUS_instance * **GENUS_createInstance** (`GC_COMPILER_NAME`, *name*, *parameter_name*, *parameter_value*, ..., ..., `GC_END_LIST`)

char *	*name*;
GENUS_parameterName	*parameter_name*;
[value type]	*parameter_value*;

This call will first attempt to create a GENUS generic component from the parameters given. Once this has completed, either by finding a component that already exists or by creating a new one, the new instance is added to that component.

The final way to call **GENUS_createInstance** corresponds to the second way to call **GENUS_createComponent** and exists because a non-variable argument list may be needed at program runtime.

GENUS_instance * **GENUS_createInstance** (`GC_PARAMETER_LIST`,*parameter_list*)

GENUS_externalParameter	*parameter_list[]*;

The first parameter in *parameter_list* must be one of the first parameters to any of the previous three calls to **GENUS_createInstance**, `GC_COMPONENT_PTR`, `GC_COMPONENT_KEY`, or `GC_PARAMETER_LIST`. Also, similarly to the previous calls, the last parameter in *parameter_list* must have name equal to `GC_END_LIST` (value is irrelevant).

The `GENUS_externalParameter` type is discussed in Section 4.3 as well as the macro **GENUS_setParameter**, which allows a parameter list to be constructed.

4.4.1 Instance Parameters

Instance specific parameters relate only to GENUS instances. Currently, we have three instance specific parameters:

- GC_INSTANCE_NAME:
  Of type char *, this parameter assigns the user defined name for this instance. As of this time there is no way to query instances by name rather than pointer.
- GC_ATTRIBUTE_PTR:
  Of type GENUS_attribute *, this parameter assigns performance metrics(area, delay, etc.) to the instance. The GC_ATTRIBUTE_LIST of the parent component is searched for this ATTRIBUTE. If found, a pointer is attached to it, otherwise this ATTRIBUTE is added to the GC_ATTRIBUTE_LIST and a pointer is established. Note that the delay value provided in this ATTRIBUTE is used for the delay models of the instance. The ATTRIBUTE and the delay model are discussed in Sections 7 and 9.2 respectively.
- GC_SHAPE_PTR:
  Of type GA_SHAPE, this parameter assigns a shape(length-width-delay tuple) to the instance. The shape-function associated with GC_ATTRIBUTE_PTR is searched for this SHAPE. If found, a pointer is attached to it, otherwise this SHAPE is added to the shape-function associated with GC_ATTRIBUTE_PTR and a pointer is established. The SHAPE is discussed in Section 7.

Appendix A shows an example that creates an instance of a component.

4.5 Deleting Instances and Components

An instance can be deleted at any time but a component can only be deleted if its instance list is empty.

```
Bool    GENUS_delete (indicator,value)
    GENUS_parameterName   indicator;
    caddr_t               value;
```

The return value indicates whether the deletion was successful or not.

The first parameter, *indicator*, can have one of three values, each specifying what the argument *value* will have.

- GC_COMPONENT_PTR indicates that *value* will hold a pointer to a previously defined GENUS component (GENUS_component *). The component will only be deleted if it contains no instances.
- GC_COMPONENT_KEY indicates that *value* will hold a component key hash string identifying a previously defined GENUS component (GENUS_componentKey). The component will only be deleted if it contains no instances.
- GC_INSTANCE_PTR indicates that *value* wil hold a pointer to an instance (GENUS_instance *).

5 Querying Information From Generators, Components, and Instances

Querying information from a GENUS generator, component or instance works on a parameter name basis. It is hierarchical in the sense that querying an instance for component or generator information, or querying a component for generator information works as expected.

The GENUS query routines also support queries based on component operations. Since parameter values may differ depending on which operation is selected, a separate routine is required to initialize a component's operation parameters. Section 5.4 describes the **GENUS_setComponentOperation** procedure and how it can be used to retrieve operation-specific information. Operation-specific parameter names are shown in Appendix C.2.

From a very general viewpoint, parameter names are type independent. This means that every query procedure must first establish the type of the parameter and then retrieve the parameter's value into a data address of the same type. However, from a more practical perspective, a parameter name may also directly convey the information about its type by its name. For instance, `GC_NUM_INPUTS` clearly must be an integer for any generator or component. Thus, parameter names do not have to exist independently of type.

The GENUS query routines handle both cases above by addressing the more general first case with two general purpose routines, and the more specific second case with a number of macro functions.

5.1 Parameter Type Query

The function **GENUS_queryParameterType** can be used to return the type symbol for any named parameter. The value returned is of the type `GENUS_parameterType`, which is discussed in detail in Section 4.1. Table 1, in the same section, shows the possible type symbols and their definitions.

GENUS_queryParameterType will return type symbol `GC_TypeUndef` on any error.

GENUS_parameterType **GENUS_queryParameterType** (*indicator*,*value*,*parameter*)
GENUS_parameterName *indicator*;
caddr_t *value*;
GENUS_parameterName *parameter*;

The first argument, *indicator*, can have one of four values, each specifying what the argument *value* will have.

- `GC_COMPILER_NAME` indicates that *value* will hold a string identifying a GENUS generic component generator. Its type is `char *`.
- `GC_COMPONENT_PTR` indicates that *value* will hold a pointer to a previously defined GENUS component. Its type is `GENUS_component *`.
- `GC_COMPONENT_KEY` indicates that *value* will hold a component key hash string identifying a previously defined GENUS component. Its type is `GENUS_componentKey`.
- `GC_INSTANCE_PTR` indicates that *value* will hold a pointer to an instance. Its type is `GENUS_instance *`.

The last argument, *parameter*, gives the parameter name. Appendix C.1 lists all of the parameter names.

5.2 Parameter Value Query

The function **GENUS_queryParameter** is the central information query routine.

Bool **GENUS_queryParameter** (*indicator*,*value*,*parameter*,*type*,*data*)

GENUS_parameterName	*indicator*;
caddr_t	*value*;
GENUS_parameterName	*parameter*;
GENUS_parameterType	*type*;
caddr_t *	*data*;

The first argument, *indicator*, can have one of four values, `GC_COMPILER_NAME`, `GC_COMPONENT_PTR`, `GC_COMPONENT_KEY`, or `GC_INSTANCE_PTR`, each specifying what the argument *value* will have, the same way as **GENUS_queryParameterType** described above. The third argument, *parameter*, gives the parameter name. The fourth argument, *type*, specifies the type of the parameter value associated with the parameter name, and the last parameter provides a storage address for the parameter value.

GENUS_queryParameter returns a boolean value indicating whether or not the call was successful. If the call was not successful, either because of bad arguments or an undefined parameter value, the data address will be set to NULL.

With these two routines, **GENUS_queryParameterType** and **GENUS_queryParameter**, any and all information can be queried from GENUS generators, components, or instances.

The following example shows how the number of inputs value could be queried from a previously defined AND component.

```
int get_num_inputs(and_c)

   GENUS_component *and_c;
{
   GENUS_parameterType  num_inputs_type;
   int                  num_inputs;
   Bool                 ret_value;

   num_inputs_type = GENUS_queryParameterType(GC_COMPONENT_PTR,
                          (caddr_t)and_c, GC_NUM_INPUTS);
   if(num_inputs_type != GC_Int){
      fprintf(stderr, "Unknown type for GC_NUM_INPUTS\n");
      return -1;
   }

   ret_value = GENUS_queryParameter(GC_COMPONENT_PTR, (caddr_t)and_c,
                                 GC_NUM_INPUTS, num_inputs_type,
                                 (caddr_t *)&num_inputs);

   if(ret_value)
      return num_inputs;
   else{
      fprintf(stderr, "Error from GENUS_queryParameter.\n");
      return -1;
   }
}
```

Notice that `and_c` in both calls is cast to `(caddr_t)`, and `&num_inputs` is cast to `(caddr_t *)`. This prevents lint or SaberC warnings.

As previously mentioned, unlike parameter `GC_NUM_INPUTS`, some of the parameters such as `GC_FUNCTION_LIST` can have different types in different scenarios. For example, `GC_FUNCTION_LIST` has type `GC_Operation` for `AND` generator, type `GC_OperationList` for `ALU` generator and type `GC_CharPtrPtr` for `MUX` generator. We need to take different actions based on its type(*GC_Operation, GC_OperationList or GC_CharPtrPtr*). The following example uses different set of instructions in order to print `GC_FUNCTION_LIST` based on the value returned by the type query.

```
void print_function_list(comp)

     GENUS_component *comp;
{
     GENUS_parameterType         type;
     Bool                        ret_value;
     GENUS_charPtrPtrType        *cptrptr;
     GENUS_operationListType     *op_list;
     char                        **cptrlist;
     GENUS_operation  op, *opl ;
     int                         length;
     register                    i, j;

     type = GENUS_queryParameterType(GC_COMPONENT_PTR, (caddr_t) comp,
                GC_FUNCTION_LIST);

     printf("Function list :\n");
     switch(type)
     {
        case GC_Operation :
             ret_value = GENUS_queryParameter(GC_COMPONENT_PTR, (caddr_t) comp,
                                 GC_FUNCTION_LIST, type, &op);
             if (ret_value)
                printf("%s\n", GENUS_operationText(op));
             else
                fprintf(stderr, "Error from GENUS_queryParameter.\n");
             break;

        case GC_OperationList :
             ret_value = GENUS_queryParameter(GC_COMPONENT_PTR, (caddr_t)comp,
                                  GC_FUNCTION_LIST, type, &op_list);
             if (ret_value)
     {
                opl = GENUS_operationListQuery(op_list, &length);
                for (i = 0; i < length; i++)
                    printf("%s, ", GENUS_operationText(opl[i]));
                printf("\n");
             }
```

```
            else
               fprintf(stderr, "Error from GENUS_queryParameter.\n");
            break;

        case GC_CharPtrPtr :
            ret_value = GENUS_queryParameter(GC_COMPONENT_PTR, (caddr_t)comp,
                                  GC_FUNCTION_LIST, type, (caddr_t)&cptrptr);
            if (ret_value)
      {
               cptrlist = GENUS_charPtrPtrQuery(cptrptr, &length);
               for (i = 0; i < length; i++)
                   printf("%s, ", (char *) cptrlist[i]);
               printf("\n");
            }
            else
               fprintf(stderr, "Error from GENUS_queryParameter.\n");
            break;

        default :
            fprintf(stderr, "Error from GENUS_queryParameter.\n");
      }
}
```

Appendix C.3 shows the list of possible types for each parameter.

5.3 Query Macros

The GENUS query macros provide ways to make generator, component, or instance specific queries in a *type specific* way.

Table 2 lists all of the query macros. The generator macros (**GCQ_Gen**<*Type*>) take a generator name `char *` and a parameter name and return a value of the type described. The component macros (**GCQ_Comp**<*Type*>) take a component pointer `GENUS_component *` and parameter name. The instance macros (**GCQ_Inst**<*Type*>) take an instance pointer `GENUS_instance *` and parameter name. (The `GC_COMPONENT_KEY` indicator type has been omitted, here.)

The following example shows how `get_num_inputs()`, from above, could be rewritten with the query macros.

```
int get_num_inputs(and_c)

   GENUS_component *and_c;
{
   return GCQ_CompInt(and_c, GC_NUM_INPUTS);
}
```

While certainly simpler, one must notice that no error checking is, or can be performed in this example. If `and_c` for some reason does not have `GC_NUM_INPUTS` defined the value returned

Type Symbol	Type returned	Generator Macros	Component Macros	Instance Macros
GC_Int	int	GCQ_GenInt	GCQ_CompInt	GCQ_InstInt
GC_Boolean	Boolean	GCQ_GenBool	GCQ_CompBool	GCQ_InstBool
GC_CharPtr	char *	GCQ_GenCharP	GCQ_CompCharP	GCQ_InstCharP
GC_ComponentKey	GENUS_componentKey		GCQ_CompKey	GCQ_InstKey
GC_Misc	GENUS_miscParameterValue	GCQ_GenMisc	GCQ_CompMisc	GCQ_InstMisc
GC_MiscList	GENUS_miscListType *	GCQ_GenMiscList	GCQ_CompMiscList	GCQ_InstMiscList
GC_Class	GENUS_class	GCQ_GenClass	GCQ_CompClass	GCQ_InstClass
GC_Operation	GENUS_operation	GCQ_GenOp	GCQ_CompOp	GCQ_InstOp
GC_OpClasses	GENUS_opClassListType *	GCQ_GenOpCls	GCQ_CompOpCls	GCQ_InstOpCls
GC_IntPtr	GENUS_intPtrType *	GCQ_GenIntP	GCQ_CompIntP	GCQ_InstIntP
GC_CharPtrPtr	GENUS_charPtrPtrType *	GCQ_GenCharPP	GCQ_CompCharPP	GCQ_InstCharPP
GC_CharPtrPtrPtr	GENUS_charPtrPtrPtrType *	GCQ_GenCharPPP	GCQ_CompCharPPP	GCQ_InstCharPPP
GC_InstanceList	GENUS_instanceListType *		GCQ_CompInstList	
GC_OperationList	GENUS_operationListType *	GCQ_GenOpList	GCQ_CompOpList	GCQ_InstOpList
GC_AttributeList	GENUS_attributeListType *		GCQ_CompAtList	GCQ_InstAtList
GC_AttributePtr	GENUS_attribute *			GCQ_InstAtP
GC_ShapePtr	GA_SHAPE *			GCQ_InstSpP

Table 2: GENUS Query Macros

will be a possibly legitimate 0. In addition, if `GC_NUM_INPUTS` is defined as `GENUS_intPtrType *`, for example (as might possibly be the case for a fictitious variable register size register file), the value returned will also be 0. However, in the latter case an error message will be generated internally.

5.4 Querying Component Information by Operator

A GENUS component may have several operations it can perform. Each operation has a specific set of inputs, outputs, and control line names associated with it. The operator specific parameter names are listed in Appendix C.2. In general, they always have the prefix `GC_OP_`. The procedure **GENUS_setOperation** is provided as a way to specify the operation to which later query routines will apply.

Boolean **GENUS_setOperation** (*indicator,value,op_type,operator*)
GENUS_parameterName *indicator*;
caddr_t *value*;
GENUS_parameterType *op_type*;
caddr_t *operator*;

The first argument, *indicator*, can have one of three values, each specifying what the argument *value* will have.

- `GC_COMPONENT_PTR` indicates that *value* contains a pointer to a GENUS component (`GENUS_component *`)
- `GC_COMPONENT_KEY` indicates that *value* holds a component key hash string identifying a previously defined GENUS component (`GENUS_componentKey`).
- `GC_INSTANCE_PTR` indicates that *value* contains a pointer to a GENUS instance (`GENUS_instance *`). In this case, the operator will be applied to the component which owns this instance.

The third argument, *op_type*, specifies the type for the operator. This is required since different components may use different types to represent operators. Currently, one of `GC_Operation`, `GC_Misc` and `GC_CharPtr` are valid choices for *op_type*.

The last argument, *operator*, specifies the desired operator, and is of the type specified by *op_type*.

The following example shows how **GENUS_setOperation** can be used on a logic unit (LU) component to set the operator `GC_RID`. A subsequent query of `GC_OP_INPUT_LIST` returns the list of inputs active for the operator `GC_RID`.

```
print_input_list_rid(lu)

   GENUS_component *lu;
{
   char                          ***input_list;
   int                           length, *height;
   GENUS_charPtrPtrPtrType       *internal_type;
   register                      i,j;

   if(!GENUS_setOperation(GC_COMPONENT_PTR, lu, GC_Operator, GC_RID))
      /* procedure will give error message */
      return;

   internal_type = GCQ_CompCharPPP(lu, GC_OP_INPUT_LIST);
   if(!internal_type)
      return;

   input_list = GENUS_charPtrPtrPtrQuery(internal_type, &length, &height);

   for(i=0; i< length; i++){
      for(j=0; j< height[i]; j++)
         printf("%s, ", input_list[i][j]);
      printf("\n");
   }
}
```

6 User Defined Components

GENUS supports the definition of user-defined components and their subsequent access through programming library calls. Although GENUS generators provide a good coverage for the currently existing libraries [JhD93s], it is not possible to provide 100% coverage with a fixed set of generators. There will always be some component that are not characterized by a GENUS generator. The user would like to define such components and use this component alongwith other GENUS components. Thus, with the supplement of user-defined components, virtually any design could be specified in GENUS environment.

Once a user-defined component is created, it behaves like any other component. The user can create and/or delete instances of user-defined components, add and retrieve performance metrics. All query (except operation based) routines are applicable to these components and their instances. Also, a user-defined component could be instantiated from the specifications provided in GENUS_LIBRARY_DIR (refer section 11.3). Finally, user-defined components can be deleted just like any other component.

A user-defined component is specified by its input and output ports, a unique (user-defined) component key, and three optional file-names:

GC_DOC_FILE : This is the name of a documentation file that can be used to describe the functionality of the component, databook information, special attributes for the component, or any other information that the user wished to associate with the user-defined component. GENUS will not parse or attempt to understand the contents of this file, but will return the file name with the appropriate query.

GC_VHDL_FILE : This is the simulation model file name for user-defined component.

GC_LOGIC_FILE : This is the logic synthesis model file name for the user-defined component.

6.1 Creating User Defined Components

A user-defined component is created by specifying component-key, ports alongwith simulation and synthesis models. The same routine for creating components, **GENUS_createComponent**, is used to create the user-defined components. Recall that there are two ways to call GENUS_createComponent.

GENUS_component * **GENUS_createComponent** (`GC_COMPILER_NAME`, *name*, *parameter_name*, *parameter_value*, ..., ..., `GC_END_LIST`)
char * *name*;
GENUS_parameterName *parameter_name*;
[value type] *parameter_value*;

GENUS_component * **GENUS_createComponent** (`GC_PARAMETERS`,*parameter_list*)
GENUS_externalParameter *parameter_list[]*;

As before, the first parameter in these calls must be GC_COMPILER_NAME. To create a user-defined component, the parameter-value for GC_COMPIELR_NAME should be "USER_DEFINED". User-defined components are grouped under "USER_DEFINED" generator. Following is the list of parameters alongwith their external and internal types for "USER_DEFINED" generator:

```
GC_EXTERNAL_PARAMETERS:      /* variables */
        (GC_COMPONENT_KEY,   GC_Required, GC_ComponentKey,%k,
         GC_NUM_INPUTS,      GC_Optional, GC_Int,         %ni,
         GC_INPUT_LIST,      GC_Optional, GC_EPortList,
                             GC_CharPtrPtrPtr,            %il,
         GC_NUM_OUTPUTS,     GC_Optional, GC_Int,         %no,
         GC_OUTPUT_LIST,     GC_Optional, GC_EPortList,
                             GC_CharPtrPtrPtr,            %ol,
         GC_NUM_INOUTS,      GC_Optional, GC_Int,         %nio,
```

```
            GC_INOUT_LIST,      GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %iol,
            GC_NUM_MISC_INPUTS, GC_Optional, GC_Int,         %nmi,
            GC_MISC_INPUT_LIST, GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %mil,
            GC_NUM_MISC_OUTPUTS,GC_Optional, GC_Int,         %nmo,
            GC_MISC_OUTPUT_LIST,GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %mol,
            GC_NUM_MISC_INOUTS, GC_Optional, GC_Int,         %nmio,
            GC_MISC_INOUT_LIST, GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %miol,
            GC_NUM_CLOCKS,      GC_Optional, GC_Int,         %ncl,
            GC_CLOCK_LIST,      GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %cll,
            GC_NUM_CONTROLS,    GC_Optional, GC_Int,         %ncn,
            GC_CONTROL_LIST,    GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %cnl,
            GC_NUM_ASYNCS,      GC_Optional, GC_Int,         %na,
            GC_ASYNC_LIST,      GC_Optional, GC_EPortList,
                                GC_CharPtrPtrPtr,            %al,
            GC_VHDL_FILE,       GC_Optional, GC_CharPtr,     %vf,
            GC_LOGIC_FILE,      GC_Optional, GC_CharPtr,     %lf,
            GC_DOC_FILE,        GC_Optional, GC_CharPtr,     %df,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}
```

Parameter GC_COMPONENT_KEY is a required parameter; type of this parameter is *GC_ComponentKey* and a string of characters should be passed as its value. Note that in user-defined components, component key is not generated internally and that user-given component key is used to store component pointers in the hash table. Thus it is user's responsibility to insure that no two different components have same key.

The other parameters for user-defined components include list of parameter name/value pairs that define port names, port widths, simulation model, synthesis model, document file and performance metric attributes. All these parameters are optional. Parameters GC_VHDL_FILE, GC_LOGIC_FILE and GC_DOC_FILE specify simulation model file, synthesis model file and document files respectively. Parameters GC_INPUT_LIST, GC_OUTPUT_LIST, ..., GC_ASYNC_LIST specify list of port name-width pairs. An individual port can be initialized with the **GENUS_setPort** macro which has the following format:

void **GENUS_setPort** (*pt*,*name*,*width*)

GENUS_port	*pt*;
char *	*name*;
int	*width*;

Following example creates a user-defined component with two inputs, an output, a clock port and a simulation model file.

```
create_blackbox()
{
   GENUS_component             *blackbox_component;
   GENUS_port                  input_list[2], output_list[1],
                               clock_list[1];
   GENUS_externalParameter     list[10];
   int                         i;

   GENUS_setPort(input_list[0], "A", 4);
   GENUS_setPort(input_list[1], "B", 4);

   GENUS_setPort(output_list[0], "C", 4);

   GENUS_setPort(clock_list[0], "CLK", 1);

   i = 0;
   GENUS_setParameter(list[i], GC_COMPILER_NAME, "USER_DEFINED"); i++;
   GENUS_setParameter(list[i], GC_COMPONENT_KEY, "BlackBox"); i++;
   GENUS_setParameter(list[i], GC_NUM_INPUTS, 2); i++;
   GENUS_setParameter(list[i], GC_INPUT_LIST, input_list);i++;
   GENUS_setParameter(list[i], GC_NUM_OUTPUTS, 1); i++;
   GENUS_setParameter(list[i], GC_OUTPUT_LIST, output_list);i++;
   GENUS_setParameter(list[i], GC_NUM_CLOCKS, 1); i++;
   GENUS_setParameter(list[i], GC_CLOCK_LIST, clock_list);i++;
   GENUS_setParameter(list[i], GC_VHDL_FILE, "BlackBox.vhdl");i++;
   GENUS_setParameter(list[i], GC_END_LIST, 0); i++;

   blackbox_component = GENUS_createComponent(GC_PARAMETER_LIST, list);
}
```

7 Performance Metrics

GENUS provides a mechanism to incorporate performance metrics such as area, delay, etc. of a component. These metrics guide the designer or the synthesis tool in making decisions regarding design trade-offs. GENUS also supports data structures to store physical design information in terms of shape functions for a component.

GENUS handles the following parameters : area, gate count, flip-flop count, delay, technology name, style of implementation, library name, name of the file containing miscellaneous information regarding the component and physical design information in terms of shape functions. The following data structures and macros are used for storing the above parameters:

```
/* Area of the design */
typedef struct _pair1 {
  long value;
  char *unit;
} area;

#define _areaValue(p)                                  ((p)->value)
#define _areaUnit(p)                                   ((p)->unit)

/* Gate count for the design */
typedef struct _pair2 {
  long value;
  char *unit;
} gate;

#define _gateValue(p)                                  ((p)->value)
#define _gateUnit(p)                                   ((p)->unit)

/* Flip-flop count for the design */
typedef struct _pair3 {
  long value;
  char *unit;
} ff;

#define _ffValue(p)                                    ((p)->value)
#define _ffUnit(p)                                     ((p)->unit)

/* Propagation delay for the design */
typedef struct _pair4 {
  float value;
  char *unit;
} delay;

#define _delayValue(p)                                 ((p)->value)
#define _delayUnit(p)                                  ((p)->unit)

/* Performance metrics for a design */
typedef struct _attribute {
  area *at_area;
  gate *at_gate;
  ff *at_ff;
  delay *at_delay;
  char *at_techname;              /* Technology name : FPGA */
  char *at_style;                 /* Style of implementation : bit-slice */
  char *at_libname;               /* Library : Xilinx 4000 */
```

```
  char  *at_filename;            /* Miscellaneous info */
  pd *at_pd;                     /* Physical design info */
} attribute, GENUS_attribute;

#define _attributeArea(p)                                 ((p)->at_area)
#define _attributeGate(p)                                 ((p)->at_gate)
#define _attributeFf(p)                                   ((p)->at_ff)
#define _attributeDelay(p)                                ((p)->at_delay)
#define _attributeTechname(p)                             ((p)->at_techname)
#define _attributeStyle(p)                                ((p)->at_style)
#define _attributeLibname(p)                              ((p)->at_libname)
#define _attributeFilename(p)                             ((p)->at_filename)
#define _attributePd(p)                                   ((p)->at_pd)
```

7.1 Incorporating performance metrics into GENUS

The following routine creates a `GC_ATTRIBUTE_PTR`, given values to its constituent fields such as `area_value`, `area_unit`, etc. The `GC_ATTRIBUTE_PTR` could then be used as one of the parameters to generate an instance of a component as mentioned in Section 4.4.

```
GENUS_attribute *      GENUS_createAttribute(
             area_value, area_unit,
             gate_value, gate_unit,
             ff_value, ff_unit,
             delay_value, delay_unit,
             techname, style,
             libname, filename, pd_info)
    long               area_value, gate_value, ff_value;
    float              delay_value;
    char *             area_unit, gate_unit, ff_unit, delay_unit,
                       techname, filename, filename, libname;
    pd                 *pd_info;
```

The parameters *area_value, gate_value, ff_value and delay_value* expects a non-negative value. If a negative value is passed for these parameters, the routine associates a NULL value to the field and it does not require the corresponding *unit* parameter.

Similarly, the following routine creates a `GC_ATRIBUTE_LIST`.

```
GENUS_attribute *      GENUS_createAttributeList(num_attributes,
             area_value, area_unit,
             gate_value, gate_unit,
             ff_value, ff_unit,
             delay_value, delay_unit,
             techname, style,
             libname, filename,
             pd_info,
```

```
                area_value, area_unit,
                ........          )

         int                num_attributes;
         long               area_value, gate_value, ff_value;
         float              delay_value;
         char *             area_unit, gate_unit, ff_unit, delay_unit,
                            techname, style, filename, libname;
         pd                 *pd_info;
```

The first parameter *num_attributes* specifies the number of the attributes to be created. The following example illustrates the use of these routines.

```
create_component_with_metrics()
{

  GENUS_operation             function_list[30];
  GENUS_component             *addsub_component;
  GENUS_instance              *addsub_instance;
  GENUS_attribute             *at;
  GENUS_externalParameter     list[10];
  int                          i;

/*  Initialize GENUS */
  if (!GENUS_initialize()) {
    printf("Some problem in initializing \n");
    exit(0);
  }

/* Set up the function list */
  i = 0;
  function_list[0] = GC_ADD;

/*  Set up the attribute list */
  at = GENUS_createAttributeList(2,     /* Number of attributes */
               /* Attribute 1 */
                    10000, "sqmicron",  /* area */
                    -1,                 /* null for gate_count */
                    -1,                 /* null for ff_count */
                    (float) 10, "ns",   /* delay */
                    "FPGA", "SRAM",    /* Technology name and Style */
                    "Xilinx4000", NULL, /* libname and filename */
                    (pd *) NULL,        /* null to pd info */

               /* Attribute 2 */
                    -1,                 /* null for area */
                    200, "2nand",       /* gate_count */
                    10, "Dff",          /* ff_count */
                    -1.0,               /* null for delay */
                    NULL, NULL,         /* Technology name and Style */
                    "lsi", "xyz.data",  /* libname and filename*/
                    (pd *) NULL);       /* null to pd info */
```

```
/*  Create an ADD_SUB component with two designs */
  GENUS_setParameter(list[i], GC_COMPILER_NAME, "ADD_SUB"); i++;
  GENUS_setParameter(list[i], GC_INPUT_WIDTH, 4); i++;
  GENUS_setParameter(list[i], GC_NUM_FUNCTIONS, 1);i++;
  GENUS_setParameter(list[i], GC_FUNCTION_LIST, function_list); i++;
  GENUS_setParameter(list[i], GC_STYLE, GC_RIPPLE_CARRY); i++;
  GENUS_setParameter(list[i], GC_NUM_ATTRIBUTES, 2); i++;
  GENUS_setParameter(list[i], GC_ATTRIBUTE_LIST, at); i++;
  GENUS_setParameter(list[i], GC_END_LIST, 0); i++;

  addsub_component = GENUS_createComponent(GC_PARAMETER_LIST, list);

  at = GENUS_createAttribute(
                  10000, "sqmicron",  /* area */
                  -1,                 /* null for gate_count */
                  -1,                 /* null for ff_count */
                  (float) 10, "ns",   /* delay */
                  NULL, NULL,         /* Technology name and Style */
                  NULL, NULL,         /* libname and filename */
                  (pd *) NULL);       /* null to pd info */

/* Create an instance with given area, delay values */
  i = 0;
  GENUS_setParameter(list[i], GC_COMPONENT_PTR, addsub_component); i++;
  GENUS_setParameter(list[i], GC_INSTANCE_NAME, "Adder1"); i++;
  GENUS_setParameter(list[i], GC_ATTRIBUTE_PTR, at); i++;
  GENUS_setParameter(list[i], GC_END_LIST, 0); i++;

  addsub_instance = GENUS_createInstance(GC_PARAMETER_LIST, list);
}
```

The second way to incorporate performance metrics into GENUS is by reading in these parameters from a file.

```
Bool            GENUS_addAttributeFile(name, component)
     char            *name;
     GENUS_component *component;
```

The argument *name* is the name of the file containing the performance metrics for the component pointed by the argument *component*. A sample input file is shown below.

```
GC_COMPILER_NAME : ADD_SUB;
GC_INPUT_WIDTH : 4;
GC_NUM_FUNCTIONS : 2;
GC_FUNCTION_LIST : GC_ADD, GC_SUB;
GC_STYLE : GC_RIPPLE_CARRY;

area_value = -1         area_unit = none
gate_value = 32         gate_unit = 2nand
ff_value = 10           ff_unit = Dff
```

Parameter name	*Type*	*Parameter value*	
		Type	Default
area_value	string constant	long int	negative number
area_unit	string constant	string	any string
gate_value	string constant	long int	negative number
gate_unit	string constant	string	any string
ff_value	string constant	long int	negative number
ff_unit	string constant	string	any string
delay_value	string constant	float	negative number
delay_unit	string constant	string	any string
techname	string constant	string	'NULL'
style	string constant	string	'NULL'
libname	string constant	string	'NULL'
filename	string constant	string	'NULL'

Table 3: Performance metrics parameters

```
delay_value = 5.68      delay_unit = ns
techname = FPGA         style = SRAM
libname = xilinx4000    filename = NULL;

area_value = -1         area_unit = none
gate_value = 57         gate_unit = 2nand
gate_value = 5          ff_unit = Dff
delay_value = 4.53      delay_unit = ns
techname = NULL         style = NULL
libname = NULL          filename = NULL;

area_value = -1         area_unit =  none
gate_value = 63         gate_unit = 2nand
ff_value = 20           ff_unit = Dff
delay_value = 4.43      delay_unit = ns
techname = NULL         style = NULL
libname = NULL          filename = NULL;
```

The first part of the file describes the component in terms of parameter values. Each line contains a parameter name and value pair followed by a semicolon. This particular input shows metrics for three designs generated for a 4-bit adder. Each design is represented by a set of twelve 3-tuples. Each tuple consists of the parameter name (area_value, area_unit, etc.), an equality sign (=) and the value of the parameter. These ten tuples are expected in the same order as is shown here. Each piece of design information ends with a semicolon(';'). Table 3 describes these parameters.

The default value for a particular entry suggests that data is not available for that entry. For example, the value '-1' for the area_value suggests that area information for the above design is missing.

The third way to incorporate the metrics is very similar to the second way. This call needs only component-pointer; the default file with the GC_COMPONENT_KEY as name is used.

```
Bool            GENUS_addAttributeKeyFile(component)
     GENUS_component *component;
```

7.2 Querying Design Metrics

GENUS provides query routines to deal with design metrics. With these routines, one can retrieve a design with a particular property. For example, the design having the best area, the design with the best gate count, the design with the best flip-flop count, the design with the best delay or the design that uses a particular library could be extracted. The following routines serve the above functions.

```
/* Get a design */
GENUS_attribute                  *GENUS_getAnAttribute(indicator, value)
     GENUS_parameterName         indicator;
     caddr_t                     value;
/* Get design with the Best Area */
GENUS_attribute                  *GENUS_getBestAreaAttribute(indicator, value)
     GENUS_parameterName         *indicator;
     caddr_t                     value;
/* Get design with the Best gate count */
GENUS_attribute                  *GENUS_getBestGateAttribute(indicator, value)
     GENUS_parameterName         *indicator;
     caddr_t                     value;
/* Get design with the Best ff count */
GENUS_attribute                  *GENUS_getBestFfAttribute(indicator, value)
     GENUS_parameterName         *indicator;
     caddr_t                     value;
/* Get design with the Best delay */
GENUS_attribute                  *GENUS_getBestDelayAttribute(indicator, value)
     GENUS_parameterName         *indicator;
     caddr_t                     value;
/* Get design with the given technology */
GENUS_attribute                  *GENUS_getLibAttribute(indicator, value,
                                                       techname)
     GENUS_parameterName         *indicator;
     caddr_t                     value;
     char                        *techname;
/* Get design with the given style of implementation*/
GENUS_attribute                  *GENUS_getLibAttribute(indicator, value,
                                                       style)
     GENUS_parameterName         *indicator;
     caddr_t                     value;
     char                        *style;
/* Get design with the given library name */
```

```
GENUS_attribute                *GENUS_getLibAttribute(indicator, value,
                                                      libname)
     GENUS_parameterName       *indicator;
     caddr_t                   value;
     char                      *libname;
```

The first argument *indicator* in these routines can have one of the three values, GC_COMPONENT_PTR, GC_COMPONENT_KEY or GC_INSTANCE_PTR, each specifying what the argument *value* will have. Once we get a particular design, different fields could be extracted using query routines discussed in Section 8. An example is shown below.

```
print_attribute_values(comp)

  GENUS_component            *comp;
{
  GENUS_attribute            *at;
  long                       areavalue, gatevalue, ffvalue;
  float                      delayvalue;
  Bool                       ret_value;
  char                       *areaunit, *gateunit, *ffunit, *delayunit,
                             *techname, *style, *libname, *filename;
  pd                         *pd_info;

/* Get the design with the best area */
  at = GENUS_getBestAreaAttribute(GC_COMPONENT_PTR, (caddr_t)comp);
  if (at)
    {
      ret_value = GENUS_attributeQuery(at, &areavalue, &areaunit,
                                       &gatevalue, &gateunit, &ffvalue,
                                       &ffunit, &delayvalue,
                                       &delayunit, &techname, &style,
                                       &libname, &filename, &pd_info);
      if (ret_value)
        {
          printf("   Attribute with the best area   \n");
          if (areavalue >= 0)
            printf("area = %ld %s\n", areavalue, areaunit);
          if (gatevalue >= 0)
            printf("gate = %ld %s\n", gatevalue, gateunit);
          if (ffvalue >= 0)
            printf("ff = %ld %s\n", ffvalue, ffunit);
          if (delayvalue >= 0)
            printf("delay = %f %s\n", delayvalue, delayunit);
          if (techname != NULL)
            printf("Technology name : %s\n", techname);
          if (style != NULL)
            printf("Style of implementation : %s\n", style);
          if (libname != NULL)
            printf("Library name : %s\n", libname);
          if (filename != NULL)
            printf("Miscellaneous file name : %s\n\n", filename);
        }
    }
```

7.3 Incorporating Physical Design Information

GENUS supports data-structures to store shape-functions for a component. A Shape-function is a list of shapes. Each shape refers to one particular implementation and is characterized by a length-width-delay tuple. The following routine builds a shape given a length, width and delay:

```
GA_SHAPE        GENUS_createShape(length, width, delay)
    float       length, width, delay;
```

The following routine creates an attribute from the shape-function in the given file *filename* and appends this attribute to the component *comp*.

```
Bool            GENUS_addShapeFile(filename, comp)
     char            *name;
     GENUS_component *component;
```

The input file contains four sections : component information, technology information, x-y-t information and x-y-t tuples. The format for these sections is described in Appendix B. The technology information is stored in the data-structure of attribute. The x-y-t information alongwith x-y-t values are stored in the physical design field of the attribute. Finally, this attribute is appended to the attribute list of the component.

The *pd* field of an attribute stores the shape function. Different fields of *pd* are extracted using query routines discussed in Section 8. An example is shown below.

```
print_pd_values(pd_info)

  pd            *pd_info;
{
  int                           num_shapes;
  char                          *lengthunit, *timeunit;
  float                         **shape_array;
  register                      i;
  Bool                          ret_value;
  if (pd_info)
  {
     ret_value = GENUS_pdQuery(pd_info, &num_shapes, &lengthunit,
                 &timeunit, &shape_array);
     if (ret_value)
     {
        printf("%d shape functions\n", num_shapes);
        printf("Length unit : %s\n", lengthunit);
        printf("Time unit : %s\n", timeunit);
        for (i = 0; i < num_shapes; i++)
            printf("%.2f %.2f %.2f\n", shape_array[i][0],

     }
  }
}
```

8 Complex Parameter Types and Routines

Certain kinds of parameter values require complex data structures to represent them. *Complex*, in this context, means that more than one variable is needed to fully describe the value's data. A list is an example of a complex data type. This section describes the various complex types, their data structures, and the routines that can be used to retrieve information from these data structures.

- `GENUS_opClassesListType` is associated strictly with the `GC_CLASSES_OP` parameter name. It is used to hold a list of type `GENUS_opClass` elements.

  GENUS_opClass * **GENUS_opClassesListQuery** (*value*,*length*)
  GENUS_opClassesListType * *value*;
  int * *length*;

  The query function is passed a `GENUS_opClassesListType` pointer and returns an array of `GENUS_opClass`. The length of the array is returned in *length*.

- `GENUS_intPtrType` is a data structure that handles an array of integers.

  int * **GENUS_intPtrQuery** (*value*,*length*)
  GENUS_intPtrType * *value*;
  int * *length*;

  The query function is passed a `GENUS_intPtrType` pointer and returns an array of integers. The length of the array is returned in *length*.

- `GENUS_intPtrPtrType` is a data structure that handles a table of integers.

  int ** **GENUS_intPtrPtrQuery** (*value*,*width*,*height*)
  GENUS_intPtrPtrType * *value*;
  int * *width*;
  int * *height*;

  The query function is passed a `GENUS_intPtrPtrType` pointer and returns a two-dimensional array of integers. The width and the height of the table is returned in *width* and *height*, respectively.

- `GENUS_charPtrPtrType` is a data structure that handles a list of character strings.

  char ** **GENUS_charPtrPtrQuery** (*value*,*length*)
  GENUS_charPtrPtrType * *value*;
  int * *length*;

  The query function is passed a `GENUS_charPtrPtrType` pointer and returns an array of character strings. The length of the array is returned in *length*.

- `GENUS_charPtrPtrPtrType` is a data structure that handles a list of variable sized lists of character strings. For example:

  ```
  "I0[0]", "I0[1]",
  "I1[0]", "I1[1]",
  "ENABLE",
  ```

  is a list of 3 lists which have 2, 2, 1 elements, respectively.

  char *** **GENUS_charPtrPtrPtrQuery** (*value*,*length*,*length_array*)
  GENUS_charPtrPtrPtrType * *value*;
  int * *length*;
  int * **length_array*;

The query function is passed a `GENUS_charPtrPtrPtrType` pointer and returns an array of variable sized lists. The size of the array is returned in *length* and the individual lengths of each list is returned by an array of integers of the same size as that returned in *length*.

- `GENUS_instanceListType` is a data structure that is associated strictly with the `GC_INSTANCE_LIST` parameter name. Is is used to retrieve the list of instances of a particular component.

  GENUS_instanceList **GENUS_instanceListQuery** (*value*,*length*)
  GENUS_instanceListType * *value*;
  int * *length*;

  The query function is passed a `GENUS_instanceListType` pointer and returns a list of instances. The length of the list is returned in *length*. The `GENUS_instanceList` type is discussed in detail in Section 8.1.

- `GENUS_operationListType` is a data structure handles of array of `GENUS_operation` elements.

  GENUS_operation * **GENUS_operationListQuery** (*value*,*length*)
  GENUS_operationListType * *value*;
  int * *length*;

  The query function is passed a `GENUS_operationListType` pointer and returns an array of `GENUS_operation` elements. The length of the array is returned in *length*.

- `GENUS_miscListType` is a data structure that handles a list of `GENUS_miscParameterValue` elements.

  GENUS_miscParameterValue * **GENUS_miscListQuery** (*value*,*length*)
  GENUS_miscListType * *value*;
  int * *length*;

  The query function is passed a `GENUS_miscListType` pointer and returns an array of `GENUS_miscParameterValue` elements. The length of the array is returned in *length*.

- `GENUS_attributeListType` is a data structure that handles a list of `GENUS_attribute` elements.

  GENUS_attribute * **GENUS_attributeListQuery** (*value*,*length*)
  GENUS_attributeListType * *value*;
  int * *length*;

  The query function is passed a `GENUS_attributeListType` pointer and returns an array of `GENUS_attribute` elements. The length of the array is returned in *length*.

8.1 The GENUS_instanceList Type

The `GENUS_instanceList` is a data structure that contains a linked list of GENUS instances. Two routines can be used to traverse the list and retrieve instances.

GENUS_instanceList **GetEachInstanceListElt** (*index*,*current*)
GENUS_instanceList *index*;
GENUS_instanceList *current*;

This function is passed an index pointer of type `GENUS_instanceList` which is initially set to the head of the instance list. Each call changes the index pointer and sets *current* to the current element. This function is primarily designed for use in a 'for' loop as will be shown in an example at the end of this section.

GENUS_instance * **GetInstanceListEltInstance** (*element*)
GENUS_instanceList *element*;

This function returns the instance in a `GENUS_instanceList` element.

The following example shows how **GetEachInstanceListElt** and **GetInstanceListElt** can be used to traverse the list of instances for a particular component.

```
void print_instances(component)

   GENUS_component      *component;
{
   GENUS_instanceListType      *instance_list_type;
   GENUS_instanceList          instance_list, index, curr;
   GENUS_instance              *instance;
   char                        *instance_name;
   int                         length;

   instance_list_type = GCQ_CompInstList(component, GC_INSTANCE_LIST);
   instance_list = GENUS_instanceListQuery(instance_list_type, &length);
   index = instance_list;

   for (curr = GetEachInstanceListElt(index, NULL);
       curr != NULL;
       curr = GetEachInstanceListElt(index, curr))
   {
        instance = GetInstanceListEltInstance(curr);
        instance_name = GCQ_InstCharP(instance, GC_INSTANCE_NAME);
        printf("instance : %s\n", instance_name);
   }
}
```

Note that *index* is used in the for-loop rather than *instance_list*. One should remember that the first argument to **GetEachInstanceListElt** is changed on each iteration.

8.2 The GENUS_attribute Type

The `GENUS_attribute` is a data structure used for storing the performance metrics for a particular design of a component. As mentioned in Section 7, this data structure consists of various fields for metrics. The following query routine extracts these parameters from a `GENUS_attribute`.

```
Bool          GENUS_attributeQuery(at_pointer, area_value, area_unit,
```

```
                                gate_value, gate_unit, ff_value,
                                ff_unit, delay_value,
                                delay_unit, filename, libname)
    GENUS_attribute *     at_pointer;
    long *                area_value, gate_value, ff_value;
    float *               delay_value;
    char *                area_unit, gate_unit, ff_unit, delay_unit,
                          filename, libname;
```

Section 7 shows an example that uses this function.

8.3 The GENUS_pd Type

The `GENUS_pd` is a data structure used for storing the physical design information (shape function) for a particular design of a component. As mentioned in Section 7, this data structure consists of various fields for metrics. The following query routine extracts these parameters from a `GENUS_attribute`.

```
Bool  GENUS_pdQuery(pd_info, num_shapes, lengthunit, timeunit, shape_array)

      pd            *p;
      int           *num_shapes;
      char          *lengthunit, *timeunit;
      float         ***shape_array;
```

Section 7 shows an example that uses this function.

9 Behavioral models

The GENUS system provides behavioral models to validate the designs that use GENUS components. The models are developed in simulatable VHDL[VHDL88]. These models are created in the directory named by environment variable `GENUS_VHDL_DIR`.. . . . These models have been tested on ZYCAD[ZYCD89] Version 1.0a VHDL Analyzer and Simulator and on Vantage[VANT90] Analysis Systems.

GENUS supports models with two-value (GC_BIT) and four-value (GC_MVL4) logic. Note that components having bidirectional ports (REG_FILE, MEMORY, etc.) and components dealing in tristate values (BUS, TRISTATE) can not be modeled with two-value logic; models for these components are available with only four-value logic. The default data-type for models is GC_BIT. Section 11 describes a routine that can modify the default data-type. Note that the data-type for the behavioral models could also be controlled using the optional parameter `GC_DATA_TYPE` associated with each generator.

The declaration packages for the four-value logic are in the directory *instdir/vhdl*. Packages *types.vhd and resMVL4_functions.vhd* contain the definition of the four-value logic data type, resolution function and basic operations for this data type. *types.vhd* is an abridged version of the package provided by ZYCAD [ZYCD89].

GENUS supports two types of behavioral models: functional and delay models.

9.1 Functional models

A functional model is associated with each component. It describes the functional relationship between the component's inputs and outputs; it does not incorporate propagation delays. Whenever a component is created, a functional model for that component is generated automatically. The generated model is stored in a file with the name given by GC_COMPONENT_KEY. This file is created in FUNCTIONAL_MODELS under the directory defined by GENUS_VHDL_DIR. The name of the functional model file can be extracted by querying the parameter GENUS_VHDL_FILE. Functional model generation feature of GENUS can be switched on/off using the global routine GENUS_setFeature described in Section 11.

Following is a VHDL Functional model generated by GENUS for a 4-bit ALU that can perform two functions: ADD and LNOT. This model uses MVL4 as its data-type.

```
---------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                    GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (GC_ADD, GC_LNOT) GC_STYLE = GC_CARRY_LOOKAHEAD
---------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity ALU_Xosd1 is
port (I0, I1 : in resMVL4_VECTOR(3 downto 0);
      ICIN : in resMVL4;
      O0 : out resMVL4_VECTOR(3 downto 0);
      O1, OCOUT, OZERO, OVF : out resMVL4;
      CADD, CLNOT : in resMVL4);
end ALU_Xosd1;

architecture ALU_Xosd1_behavior of ALU_Xosd1 is
begin
     process (I0, I1, ICIN, CADD, CLNOT)
          variable ctrl_vec : resMVL4_VECTOR(1 downto 0);
          variable temp : resMVL4_VECTOR(4 downto 0);
          variable temp1 : resMVL4_VECTOR(3 downto 0);
          variable temp2 : resMVL4_VECTOR(3 downto 0);
          variable zero : resMVL4_VECTOR(3 downto 0);
          variable ctrl_int :  integer;
          begin
               ctrl_vec := CADD & CLNOT;
               temp1 := I0;
               temp2 := I1;
               OVF <= '0';
               ctrl_int := CRL_INTVAL(ctrl_vec);
               case ctrl_int is
                      when 0 => temp := CARRY_ADD(temp1, temp2);
                                if (ICIN = '1') then
                                   temp := INC(temp);
                                end if;
                                OCOUT <= temp(4);
                                O1 <= '0';
                      when 1 => temp(3 downto 0) := ONES_COMP (temp1);
                                OCOUT <= '0';
```

```
                                O1 <= '0';
                          when others => null;
                end case;

                O0 <= temp(3 downto 0);
                zero := temp(3 downto 0);
                if (B2I(zero) = 0) then
                     OZERO <= '1';
                else
                     OZERO <= '0';
                end if;
-- Checking overflow
                if (((temp1(3) = '1') and (temp2(3) = '1')
                     and (temp(3) = '0'))
                     or ((temp1(3) = '0') and (temp2(3) = '0')
                     and (temp(3) = '1'))) then
                          OVF <= '1';
                else
                          OVF <= '0';
                end if;
           end process;
end ALU_Xosd1_behavior;

configuration ALU_Xosd1_config of ALU_Xosd1 is
     for ALU_Xosd1_behavior
     end for;
end ALU_Xosd1_config;
```

Appendix E contains examples of the functional models, one for each generator.

9.2 Delay models

A delay model is associated with an instance component. It describes the functional relationship between the component's inputs and outputs alongwith propagation delay. Note that the delay model generator uses the delay provided by the parameter `GC_ATTRIBUTE_PTR` while creating an instance of a component. Whenever an instance is created, a delay model for that component instance is generated automatically. The model is stored in a file with the name given by `GC_INSTANCE_NAME`. This file is created in `DELAY_MODELS` under the directory defined by `GENUS_VHDL_DIR`. Delay model generators can be switched on/off using the global routine `GENSU_setFeature` discussed in Section 11.

Currently, GENUS supports a simple timing model. For combinational components, the model assumes that the propagation delay is the maximum pin-to-pin delay among all input-output pin pairs. For sequential components, the delay value is given by the maximum of the three delays: clock to output, data_input to output and asynchronous control to output.

Following is a VHDL delay model generated by GENUS for an instance of 4-bit `ALU` with two functions (ADD and LNOT) and a propagation delay of 10 ns. This model uses two-value logic as its data-type.

```
----------------------ATTRIBUTES--------------------------
--   area = 100 sqmicron
--   delay = 10.00 ns
--   Library name = lsi
---------------------------------------------------------

use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity Alu1 is
port (I0, I1 : in BIT_VECTOR(3 downto 0);
      ICIN : in BIT;
      CADD, CLNOT : in BIT;
      O0 : out BIT_VECTOR(3 downto 0);
      O1, OCOUT, OZERO, OVF : out BIT);
end Alu1;

architecture Alu1_behavior of Alu1 is
begin
     process (I0, I1, ICIN, CADD, CLNOT)
          variable ctrl_vec : BIT_VECTOR(1 downto 0);
          variable temp : BIT_VECTOR(4 downto 0);
          variable temp1 : BIT_VECTOR(3 downto 0);
          variable temp2 : BIT_VECTOR(3 downto 0);
          variable zero : BIT_VECTOR(3 downto 0);
          variable ctrl_int :  integer;
          constant delay : time := 10.00 ns;
          begin
               ctrl_vec := CADD & CLNOT;
               temp1 := I0;
               temp2 := I1;
               OVF <= transport '0' after delay;
               ctrl_int := CRL_INTVAL(ctrl_vec);
               case ctrl_int is
                      when 0 => temp := CARRY_ADD(temp1, temp2);
                                if (ICIN = '1') then
                                   temp := INC(temp);
                                end if;
                                OCOUT <= transport temp(4) after delay;
                                O1 <= transport '0'after delay;
                      when 1 => temp(3 downto 0) := ONES_COMP (temp1);
                                OCOUT <= transport '0' after delay;
                                O1 <= transport '0' after delay;
                      when others => null;
               end case;

               O0 <= transport temp(3 downto 0) after delay;
               zero := temp(3 downto 0);
               if (B2I(zero) = 0) then
                    OZERO <= transport '1' after delay;
               else
                    OZERO <= transport '0' after delay;
               end if;
-- Checking overflow
```

```
            if (((temp1(3) = '1') and (temp2(3) = '1') and (temp(3) = '0'))
                 or ((temp1(3) = '0') and (temp2(3) = '0') and
                    (temp(3) = '1'))) then
                      OVF <= transport '1' after delay;
            else
                      OVF <= transport '0' after delay;
            end if;
        end process;
end Alu1_behavior;

configuration Alu1_config of Alu1 is
     for Alu1_behavior
     end for;
end Alu1_config;
```

10 Synthesis Models and Generic Gate Count

Synthesis models are also generated automatically whenever a component is created. A synthesis model consists of logic equations for each of the component's outputs, and can be used directly as input to logic synthesis tools. A component can be realized by mapping its logic equations to a specific target technology. These logic equations are stored in a file with the name given by `GC_COMPONENT_KEY`. The logic equation files are generated in the directory named by environment variable `GENUS_LOGIC_DIR` . Synthesis model file name can be extracted by querying the parameter `GENUS_LOGIC_FILE`. The synthesis model generators can be switched on/off using the global routine `GENUS_setFeature` described in Section 11.

Logic equations for the combinatorial components are in MISII [BRWa87] format. For sequential components, we employ a hierarchical VHDL netlist format that uses a combination of MISII and VHDL. At the top level, we have a netlist of storage elements and combinational blocks. At the leaf level, these combinational blocks are described using MISII format. Appendix F includes synthesis models for some sample sequential components in hierarchical netlist format.

The following is an example of a synthesis model generated by GENUS for 4-bit `ALU` that can perform two functions: ADD and LNOT.

```
INORDER = CADD[0] CLNOT[0] I0[0] I0[1] I0[2] I0[3] I1[0] I1[1]
I1[2] I1[3] ICIN[0];
OUTORDER = O0[0] O0[1] O0[2] O0[3] OCOUT[0] O1[0] OZERO[0] OVF[0];

C0 = (CADD[0]*ICIN[0]);

X[0] = (CADD[0]*I0[0]);
X[1] = (CADD[0]*I0[1]);
X[2] = (CADD[0]*I0[2]);
X[3] = (CADD[0]*I0[3]);
Y[0] = (CADD[0]*I1[0]);
Y[1] = (CADD[0]*I1[1]);
Y[2] = (CADD[0]*I1[2]);
Y[3] = (CADD[0]*I1[3]);

AB[0]   = I0[0]  * I1[0];
```

```
ABN[0]  = I0[0]  * !I1[0];
ANB[0]  = !I0[0] * I1[0];
ANBN[0] = !I0[0] * !I1[0];

AB[1]   = I0[1]  * I1[1];
ABN[1]  = I0[1]  * !I1[1];
ANB[1]  = !I0[1] * I1[1];
ANBN[1] = !I0[1] * !I1[1];

AB[2]   = I0[2]  * I1[2];
ABN[2]  = I0[2]  * !I1[2];
ANB[2]  = !I0[2] * I1[2];
ANBN[2] = !I0[2] * !I1[2];

AB[3]   = I0[3]  * I1[3];
ABN[3]  = I0[3]  * !I1[3];
ANB[3]  = !I0[3] * I1[3];
ANBN[3] = !I0[3] * !I1[3];

ANBAND[0] = ANB[0] * CLNOT[0];
ANBNAND[0] = ANBN[0] * CLNOT[0];

ZL[0] = ANBAND[0] + ANBNAND[0];

ANBAND[1] = ANB[1] * CLNOT[0];
ANBNAND[1] = ANBN[1] * CLNOT[0];

ZL[1] = ANBAND[1] + ANBNAND[1];

ANBAND[2] = ANB[2] * CLNOT[0];
ANBNAND[2] = ANBN[2] * CLNOT[0];

ZL[2] = ANBAND[2] + ANBNAND[2];

ANBAND[3] = ANB[3] * CLNOT[0];
ANBNAND[3] = ANBN[3] * CLNOT[0];

ZL[3] = ANBAND[3] + ANBNAND[3];

G0[0] = X[0] * Y[0];
P0[0] = X[0] + Y[0];
G0[1] = X[1] * Y[1];
P0[1] = X[1] + Y[1];
G0[2] = X[2] * Y[2];
P0[2] = X[2] + Y[2];
G0[3] = X[3] * Y[3];
P0[3] = X[3] + Y[3];

P1[1] = P0[0] * P0[1];
G1[1] = P0[1] * G0[0] + G0[1];
P1[3] = P0[2] * P0[3];
G1[3] = P0[3] * G0[2] + G0[3];

P2[2] = P1[1] * P0[2];
```

```
G2[2] = P0[2] * G1[1] + G0[2];
P2[3] = P1[1] * P1[3];
G2[3] = P1[3] * G1[1] + G1[3];

C1 = G0[0] + P0[0] * C0;
C2 = G1[1] + P1[1] * C0;
C3 = G2[2] + P2[2] * C0;
C4 = G2[3] + P2[3] * C0;

Z[0] = C0 ^ X[0] ^ Y[0];
Z[1] = C1 ^ X[1] ^ Y[1];
Z[2] = C2 ^ X[2] ^ Y[2];
Z[3] = C3 ^ X[3] ^ Y[3];

O1[0] = (0);
OCOUT[0] = C4;
OVF[0] = C4 ^ C3;
MS = 0 + CADD[0];

Z2[0] = (ZL[0]*!MS) + (Z[0]*MS);
Z2[1] = (ZL[1]*!MS) + (Z[1]*MS);
Z2[2] = (ZL[2]*!MS) + (Z[2]*MS);
Z2[3] = (ZL[3]*!MS) + (Z[3]*MS);

O0[0] = Z2[0];
O0[1] = Z2[1];
O0[2] = Z2[2];
O0[3] = Z2[3];

OZERO[0] = !(Z2[0] + Z2[1] + Z2[2] + Z2[3]);
```

Appendix F contains examples of synthesis models for other components.

10.1 Generic Gate Count

As mentioned before, GENUS provides generic gate counts for a component. Generic gate count includes an approximate count of equivalent 2-input gates and flip-flops. For combinational components, flip-flop count is zero. Note that the gate counts are generated from a *synthesis models* and hence the synthesis model generators should be active to get gate counts. Also, gate count generation feature can be switched on/off using the global routine `GENUS_setFeature` discussed in Section 11.

These gate counts are stored as an `ATTRIBUTE` with "GENERIC" as `libname`. This `ATTRIBUTE` can then be extracted using the following routine described in Section 7:

```
print_gate_count(comp)
    GENUS_component  *comp;
{
    GENUS_attribute  *at;
    long             areavalue, gatevalue, ffvalue;
    float            delayvalue;
    Bool             ret_value;
```

```
    char              *areaunit, *gateunit, *ffunit, *delayunit,
                      *techname, *style, *libname, *filname;
    GENUS_pd          *pd_info;

/* Get the attribute with generic gate count */
  at = GENUS_getLibAttribute(GC_COMPONENT_PTR, (caddr_t)comp, "generic");
  if (at)
  {
     ret_value = GENUS_attributeQuery(at, &areavalue, &areaunit,
                                      &gatevalue, &gateunit, &ffvalue,
                                      &ffunit, &delayvalue,
                                      &delayunit, &techname, &style,
                                      &libname, &filname, &pd_info);
     if (ret_value)
     {
          printf("   Attribute with generic gate count   \n");
          if (gatevalue >= 0)
            printf("       gate = %ld %s\n", gatevalue, gateunit);
          if (ffvalue >= 0)
            printf("       ff = %ld %s\n", ffvalue, ffunit);
     }
  }
}
```

Following is the output of the above routine when applied on a 4-bit ALU with two functions (ADD and LNOT).

```
Attribute with generic gate count
        gate = 129 2input
        ff = 0 DFF
```

11 Miscellaneous Routines and Features

This section lists few miscellaneous routines. Each of these routines has usage in a specific domain.

11.1 Selecting Modules in GENUS

As mentioned before, each module or *feature* shown in Figure 1 can be switched on or off on the need basis. Specifically, we can control these features: Functional model generators, Delay model generators, Synthesis model generators, Gate count generators and Technology library routines. The following routine sets the flags for the these five features.

```
Bool          GENUS_setFeature(feature_name, value)
   GENUS_feature  feature_name;
   Bool           value;
```

The first argument is of type GENUS_feature. Second argument could have either of two values, *True or False*, signifying the activation or deactivation of the corresponding feature.

The return value indicates whether the operation was successful or not. This routine could activated anytime during the interaction with GENUS.

The following table lists values that GENUS_features could take:

GENUS_feature value	**Corresponding feature**	*Default value*
GF_FUNCTIONAL_MODEL	Functional model generators	False
GF_DELAY_MODEL	Delay model generators	False
GF_SYNTHESIS_MODEL	Synthesis model generators	False
GF_GATE_COUNT	Gate cont generators	False
GF_COMPONENT_LIBRARY	Tech. Lib. attribute routines	False

Following example deactivates the Functional model generators, while activating the synthesis model generators.

```
Bool     func_ret_value, synth_ret_value;
func_ret_value = GENUS_setFeature(GF_FUNCTIONAL_MODEL, False);
synth_ret_value = GENUS_setFeature(GF_SYNTHESIS_MODEL, True);
```

11.2 Setting the Default Data-type for Behavioral Models

Following routine sets the default data-type to be used for behavioral (functional and delay) models.

```
Bool       GENUS_setDataType(data_type)
   GC_MISC   data_type;
```

Parameter *data_type* could have either of these two values: GC_BIT or GC_MVL4. GC_BIT signifies the two-value logic whereas GC_MVL4 means four-value logic.

11.3 Instantiating Components from a Technology Library

GENUS provides a mechanism to instantiate components alongwith attribute values (area, delay, etc.) specific to a technology library. At the time of initialization, GENUS checks for the environment variable `GENUS_LIBRARY_DIR`. If it is set, it reads in all the files from this directory and creates a component per file. For each technology specific component in the library, GENUS expects a file. These files contain parameter and attribute values for a component. Format for this file is described in Section 7. This feature of creating technology specific components could be switched on/off with the global routine `GENUS_setFeature` discussed above.

11.4 Printing External Parameters for Unit List in BIF

The following routine prints the list of external parameters and their value for a given component.

```
Bool           GENUS_printExternParameter(indicator, value, fp)
     GENUS_parameterName    indicator;
     caddr_t                value;
     FILE                   *fp;
```

The first argument, *indicator*, can have one of three values, each specifying what the argument *value* will have.

- GC_COMPONENT_PTR indicates that *value* contains a pointer to a GENUS component (GENUS_component *)
- GC_COMPONENT_KEY indicates that *value* holds a component key hash string identifying a previously defined GENUS component (GENUS_componentKey).
- GC_INSTANCE_PTR indicates that *value* contains a pointer to a GENUS instance (GENUS_instance *). In this case, the operator will be applied to the component which owns this instance.

Following is a sample output generated by the above routine.

```
GENERATOR: ALU;
PARAMETERS {
    'GC_INPUT_WIDTH:     4';
    'GC_NUM_FUNCTIONS:     2';
    'GC_FUNCTION_LIST:    (GC_ADD,GC_LNOT)';
    'GC_STYLE:     GC_CARRY_LOOKAHEAD';
}
```

11.5 Extracting Component Pointer from its Key

The following routine returns the component pointer given its key.

```
Bool            GENUS_findComponentByKey(key)
     char                    *key;
```

12 Acknowledgements

We thank Tedd Hadley for his help with the GENUS system. This work was partially supported by the Semiconductor Research Corporation under grant 93-DJ-146 and by the National Science Foundation under grant MIP-9009239.

References

[BRWa87] R. K. Brayton, A. V. R. Rudell and A. Wang, "MIS: A Multiple level logic optimization system," *IEEE Transactions on Computer-Aided Design,* November 1987.

[DuJh92] Nikil D. Dutt and Pradip K. Jha, "GENUS: A Generic Component Library for High Level Synthesis," *Technical Report 92-XX, University of California at Irvine,* February 1992.

[JhD93s] Pradip K. Jha, Nikil D. Dutt and Daniel D. Gajski, "An Evaluative Study of RT Component Libraries," *Technical Report 93-11, University of California at Irvine,* March 1993.

[JhDu93] Pradip K. Jha and Nikil D. Dutt, "Rapid Estimation for Parameterized Components in High-Level Synthesis," *IEEE Transactions on VLSI Systems,* pp296-303, September 1993.

[DuJh94] Nikil D. Dutt and Pradip K. Jha, "RT Component Sets for High-Level Design Applications," *To appear in International Journal of VLSI Design and Technical Report 94-27, University of California at Irvine, 1994.*

[LiGa88] J. S. Lis and D. D. Gajski, "Synthesis from VHDL," *Proc. IEEE Int. Conf. on Computer Design,* pp378-381, 1988.

[RaGa91] L. Ramachandran and D. D. Gajski, "An Algorithm for Component Selection in Performance Optimized Scheduling," *IEEE Int. Conf. on Computer-Aided Design,* pp92-95, November 1991.

[VHDL88] "IEEE Standard VHDL Language Reference Manual," 1988.

[ZYCD89] "ZYCAD System VHDL User's Manual," 1989.

[VANT90] "Vantage Analysis Systems: User's Guide," 1990.

A Tutorial Examples

This section presents two complete C programs that illustrate the use of GENUS. The first example covers a simple scenario in which a combinational component is created and few queries regarding ports and operations are made. The second program illustrates various features of GENUS. Specifically, it instantiates user-defined and genus components from a library along with design metrics. It then creates an instance of a sequential component, along with different models and gate-counts. This example also illustrates the use of attributes(design metrics) and shape-functions.

A.1 Example 1: Basic Interaction with GENUS

```
/*  This program, after initializing GENUS, creates a 4-bit ALU that can
    perform two functions: (GC_ADD, GC_LNOT). Next, it makes port-specific
    queries regarding GC_INPUT_LIST, GC_NUM_OUPUTS, GC_NUM_CONTROLS and
    GC_OUTPUT_WIDTH. The program then performs operation-specific queries
    for GC_ADD. Finally, it deletes the ALU component.*/

#include <stdio.h>
#include "GENUS.h"

main()
{
   GENUS_component          *alu_component;
   GENUS_externalParameter  list[6];
   GENUS_charPtrPtrPtrType  *internal_type;
   GENUS_operation          function_list[2];
   char                     *op_control, ***input_list;
   int                      num_outputs, num_controls, output_width,
                            length, *height, j, i;
   Bool                     init_value, ret_value;

   init_value =  GENUS_initialize();
   if (!init_value)
     {
       printf("Some problem in initializing the GENUS\n");
       exit(0);
     }

/* Set the function list */
   function_list[0] = GC_ADD;
   function_list[1] = GC_LNOT;

/* Set parameters for the ALU component */
   i  = 0;
   GENUS_setParameter(list[i], GC_COMPILER_NAME, "ALU"); i++;
   GENUS_setParameter(list[i], GC_INPUT_WIDTH, 4); i++;
   GENUS_setParameter(list[i], GC_NUM_FUNCTIONS, 2);i++;
   GENUS_setParameter(list[i], GC_FUNCTION_LIST, function_list); i++;
   GENUS_setParameter(list[i], GC_STYLE, GC_CARRY_LOOKAHEAD); i++;
   GENUS_setParameter(list[i], GC_END_LIST, 0); i++;
```

```
   alu_component = GENUS_createComponent(GC_PARAMETER_LIST, list);

   if(!alu_component)
     {
       printf("Some problem in creating an ALU component\n");
       exit(0);
     }

/* Query input port names */
   internal_type = GCQ_CompCharPPP(alu_component, GC_INPUT_LIST);
   if (!internal_type)
     return;

   input_list = GENUS_charPtrPtrPtrQuery(internal_type, &length, &height);

   printf("Input list :\n");
   for (i=0; i < length; i++)
     {
       for (j=0; j < height[i]; j++)
           printf("%s, ", input_list[i][j]);
       printf("\n");
     }

   num_outputs =  GCQ_CompInt(alu_component, GC_NUM_OUTPUTS);
   printf("Number of outputs : %d\n", num_outputs);

   num_controls =  GCQ_CompInt(alu_component, GC_NUM_CONTROLS);
   printf("Number of controls : %d\n", num_controls);

   output_width =  GCQ_CompInt(alu_component, GC_OUTPUT_WIDTH);
   printf("Output width : %d\n", output_width);

/* Operation specific queries*/
   if (!GENUS_setOperation(GC_COMPONENT_PTR, (void *)alu_component, GC_Operation,
   (void *)GC_ADD))
    {
      printf("Some problem in setting GC_ADD operation\n");
      exit(0);
    }

   printf("\nParameters specific to GC_ADD operation\n");

   i = GCQ_CompInt(alu_component, GC_OP_NUM_INPUTS);
   printf("OP_NUM_INPUTS : %d\n", i);

   internal_type = GCQ_CompCharPPP(alu_component, GC_OP_INPUT_LIST);
   if (!internal_type)
     {
       printf("Some problem in retrieving GC_OP_INPUT_LIST\n");
       exit(0);
     }
   input_list = GENUS_charPtrPtrPtrQuery(internal_type, &length, &height);

   printf("GC_OP_INPUT_LIST : \n");
```

```
   for (i = 0; i < length; i++)
     {
       for (j = 0; j < height[i]; j++)
           printf("%s, ", input_list[i][j]);
       printf("\n");
     }

   i = GCQ_CompInt(alu_component, GC_OP_NUM_OUTPUTS);
   printf("OP_NUM_OUTPUTS : %d\n", i);

   internal_type = GCQ_CompCharPPP(alu_component, GC_OP_OUTPUT_LIST);
   if  (!internal_type)
     {
       printf("Some problem in retrieving GC_OP_OUTPUT_LIST\n");
       exit(0);
     }
   input_list = GENUS_charPtrPtrPtrQuery(internal_type, &length, &height);

   printf("GC_OP_OUTPUT_LIST : \n");
   for (i = 0; i < length; i++)
     {
       for (j = 0; j < height[i]; j++)
           printf("%s, ", input_list[i][j]);
       printf("\n");
     }

   i = GCQ_CompInt(alu_component, GC_OP_NUM_CONTROLS);
   printf("GC_OP_NUM_CONTROLS : %d\n", i);

   op_control =  GCQ_CompCharP(alu_component, GC_OP_CONTROL_LIST);
   printf("OP_CONTROL_LIST : %s\n", op_control);

/* Delete the component */
   ret_value = GENUS_delete(GC_COMPONENT_PTR, alu_component);
}
```

The output generated by this program is shown below.

```
Input list :
I0[0], I0[1], I0[2], I0[3],
I1[0], I1[1], I1[2], I1[3],
ICIN[0],
Number of outputs : 5
Number of controls : 2
Output width : 4

Parameters specific to GC_ADD operation
OP_NUM_INPUTS : 3
GC_OP_INPUT_LIST :
I0[0], I0[1], I0[2], I0[3],
I1[0], I1[1], I1[2], I1[3],
```

```
ICIN[0],
OP_NUM_OUTPUTS : 4
GC_OP_OUTPUT_LIST :
OO[0], OO[1], OO[2], OO[3],
OCOUT[0],
OZERO[0],
OVF[0],
GC_OP_NUM_CONTROLS : 1
OP_CONTROL_LIST : CADD
```

A.2 Example 2: Model Generation and Attribute Queries in GENUS

```
/*  This program first switches on all the features, namely functional model
    generators, delay model generators, synthesis model generators, gate-count
    generators and technology library routines. Since the technology routines
    are enabled, at the time of GENUS initialization, all the components
    available in the directory specified by the environment variable
    GENUS_LIBRARY_DIR are created alongwith the design metrics. Specifically,
    for this example, we have these three files (each specifying a component),
    in the directory:

      ADD_SUB.data

           GC_COMPILER_NAME : ADD_SUB;
           GC_INPUT_WIDTH : 4;
           GC_NUM_FUNCTIONS : 2;
           GC_FUNCTION_LIST : GC_ADD, GC_SUB;
           GC_STYLE : GC_RIPPLE_CARRY;

           area_value = -1         area_unit = none
           gate_value = 32         gate_unit = 2nand
           ff_value = 10           ff_unit = Dff
           delay_value = 5.68      delay_unit = ns
           techname = FPGA         style = bit_slice
           libname = mcnc          filename = /fhjh/gdj/xyz.data;

           area_value = -1         area_unit = none
           gate_value = 57         gate_unit = 2nand
           ff_value = 5            ff_unit = Dff
           delay_value = 4.53      delay_unit = ns
           techname = NULL         style = NULL
           libname = NULL          filename = NULL;

      CONCAT.data

           GC_COMPILER_NAME : CONCAT;
           GC_NUM_INPUTS : 5;
           GC_INPUT_WIDTH_LIST : 5, 6, 2, 3, 4;

           area_value = 500        area_unit =  ksdhfk
           gate_value = 879        gate_unit = 2nand
           ff_value = -1           ff_unit = does_not_matter
           delay_value = 20.00    delay_unit = ns
           techname = FPGA         style = bit_slice
           libname = mcnc          filename = /fhjh/gdj/xyz.data;

      Blackbox.data

           GC_COMPILER_NAME : USER_DEFINED;
           GC_COMPONENT_KEY : Blackbox;
           GC_NUM_INPUTS: 2;
           GC_INPUT_LIST : (A, 4), (B, 2);
           GC_NUM_OUTPUTS: 1;
           GC_OUTPUT_LIST : (C, 4);
```

```
          GC_NUM_CONTROLS: 2;
          GC_CONTROL_LIST : (CA, 1), (CB, 1);
          GC_VHDL_FILE: /xyxz/kdjf.data;
          GC_DOC_FILE: lpr.data;

          area_value = 500        area_unit =  ksdhfk
          gate_value = 879        gate_unit = 2nand
          ff_value = 20           ff_unit = dff
          delay_value = 20.00     delay_unit = ns
          techname = FPGA         style = BIT_SLICE
          libname = Xilinx4000    filename = /fhjh/gdj/xyz.data;

    Note that the ADD_SUB component has two sets of design metrics, the CONCAT
    and Blackbox component have single set of metrics. Blackbox is a
    user-defined component, in which user has specified the component key,
    port names alongwith width, simulation and document files.The program then
    creates a 4-bit shift-register component that can perform three functions:
    GC_LOAD, GC_SHR and GC_SCAN.

    The program, then, generates an instance, "REG1", of the above shift-register
    component with an area = 10000 sqmicron, delay = 10 ns and a shape given
    by the tuple (30.0, 30.0, 10.0). It then queries
    for the GC_INSTANCE_NAME, GC_COMPONENT_KEY, GC_VHDL_FILE and GC_LOGIC_FILE.
    Note that the GC_VHDL_FILE and GC_LOGIC_FILE are located in the directories
    specified by the environment variables GENUS_VHDL_DIR and GENUS_LOGIC_DIR
    respectively and these files store functional and synthesis models
    respectively for the above shift-register component. Also, the delay model
    is stored in a file specified by GC_INSTANCE_NAME, i.e. "REG1".

    The program then extracts the gate-count for the shift-register component.
    Finally, it deletes the instance and the component. */

#include <stdio.h>
#include "GENUS.h"

main()
{
   GENUS_component          *reg_component;
   GENUS_externalParameter  list[8];
   GENUS_instance           *reg_instance;
   GENUS_componentKey       comp_key;
   GENUS_operation          function_list[3];
   GENUS_attribute          *at;
   GENUS_pd                 *pd_info;
   GA_SHAPE                 shape;
   char                     *inst_name, *areaunit, *gateunit,
                            *ffunit, *delayunit, *techname, *style,
                            *filname, *libname, *file_name;
   int                      num_outputs, i;
   Bool                     init_value, ret_value;
   long                     areavalue, gatevalue, ffvalue;
   float                    delayvalue;

/* Switch on all the features, namely, functional model generators,
```

```
   Delay model generators, Synthesis model generators, Technology library
   routines and Gate-count generators. */
   ret_value = GENUS_setFeature(GF_FUNCTIONAL_MODEL, True);
   ret_value = GENUS_setFeature(GF_DELAY_MODEL, True);
   ret_value = GENUS_setFeature(GF_SYNTHESIS_MODEL, True);
   ret_value = GENUS_setFeature(GF_COMPONENT_LIBRARY, True);
   ret_value = GENUS_setFeature(GF_GATE_COUNT, True);

   init_value =  GENUS_initialize();
   if (!init_value)
     {
       printf("Some problem in initializing the GENUS\n");
       exit(0);
     }

/* Set the function list */
   function_list[0] = GC_LOAD;
   function_list[1] = GC_SHR;
   function_list[2] = GC_SCAN;

/* Set parameters for the ALU component */
   i  = 0;
   GENUS_setParameter(list[i], GC_COMPILER_NAME, "REGISTER"); i++;
   GENUS_setParameter(list[i], GC_INPUT_WIDTH, 4); i++;
   GENUS_setParameter(list[i], GC_NUM_FUNCTIONS, 3);i++;
   GENUS_setParameter(list[i], GC_FUNCTION_LIST, function_list); i++;
   GENUS_setParameter(list[i], GC_TYPE, GC_D_FF); i++;
   GENUS_setParameter(list[i], GC_SET_VALUE, 10); i++;
   GENUS_setParameter(list[i], GC_INVERT_OUT, True); i++;
   GENUS_setParameter(list[i], GC_END_LIST, 0); i++;

   reg_component = GENUS_createComponent(GC_PARAMETER_LIST, list);
   if(!reg_component)
     {
       printf("Some problem in creating an REGISTER component\n");
       exit(0);
     }

/* Create an Attribute for the instance */
   at = GENUS_createAttribute(
                  10000, "sqmicron",  /* area */
                  -1,                 /* null for gate_count */
                  -1,                 /* null for ff_count */
                  (float) 10, "ns",   /* delay */
                  NULL, NULL,         /* techname and style */
                  NULL, NULL,         /* libname and filename */
                  (GENUS_pd *) NULL); /* physical design info */

   shape = GENUS_createShape((float) 30.00, (float) 30.00, (float) 10.00);
   i  = 0;
   GENUS_setParameter(list[i], GC_COMPONENT_PTR, reg_component); i++;
   GENUS_setParameter(list[i], GC_INSTANCE_NAME, "REG1"); i++;
   GENUS_setParameter(list[i], GC_ATTRIBUTE_PTR, at); i++;
   GENUS_setParameter(list[i], GC_SHAPE_PTR, shape); i++;
```

```
   GENUS_setParameter(list[i], GC_END_LIST, 0); i++;

/* Create an alu instance */
   reg_instance = GENUS_createInstance(GC_PARAMETER_LIST, list);

/* Simple queries */
   inst_name  = GCQ_InstCharP(reg_instance, GC_INSTANCE_NAME);
   printf("Instance name : %s\n", inst_name);

   comp_key  = GCQ_InstKey(reg_instance, GC_COMPONENT_KEY);
   printf("Component Key : %s\n", comp_key);

/* Queries for filenames of functional and synthesis models */
   file_name = GCQ_InstCharP(reg_instance, GC_VHDL_FILE);
   printf("Functional model file name : %s\n", file_name);

   file_name = GCQ_InstCharP(reg_instance, GC_LOGIC_FILE);
   printf("Synthesis model file name : %s\n", file_name);

/* Retrieve gate counts for the component */
   at = GENUS_getLibAttribute(GC_COMPONENT_PTR, reg_component, "GENERIC");

   if (at)
   {
      ret_value = GENUS_attributeQuery(at, &areavalue, &areaunit,
                                       &gatevalue, &gateunit, &ffvalue,
                                       &ffunit, &delayvalue,
                                       &delayunit, &techname, &style,
                                       &libname, &filname, &pd_info);
      if (ret_value)
      {
          printf("\n  The FF and gate counts are \n");
          if (gatevalue >= 0)
            printf("Gate-count = %ld %s\n", gatevalue, gateunit);
          if (ffvalue >= 0)
            printf("FF-count = %ld %s\n", ffvalue, ffunit);
      }
   }

/* Delete the instance and the component */
   ret_value = GENUS_delete(GC_INSTANCE_PTR, reg_instance);
   ret_value = GENUS_delete(GC_COMPONENT_PTR, reg_component);
}
```

Following is the output generated by this program. Note that although the functional model and synthesis model file names are same (component keyname), they are created in different directories and that they are followed by different extensions (.vhdl and .eqn).

```
Instance name : REG1
Component Key : REGISTER_41T1D4K381B3
Functional model file name : REGISTER_41T1D4K381B3
Synthesis model file name : REGISTER_41T1D4K381B3

  The FF and gate counts are
```

```
Gate-count = 32 2input
FF-count = 4 DFF
```

B Format for Storing Shape Functions

The shape function file contains four sections : component information, technology information, x-y-t information and x-y-t tuples. The technology information are stored in the data-structure of attribute. The x-y-t information alongwith x-y-t values are stored in the physical design field of the attribute. Finally, this attribute is appended to the attribute list of the component.

Each section in this file consists of a header line followed by a list of field name-value pairs, one per line. The header line begins with a "Each field line consists of a field name, a separator ("COLON"), field value and end marker ("SEMICOLON") in this order.

Field names in component information section are GENUS names with "GC_" prefix. Field values could be a single word or a list of words separated by comma delimiter.

The technology information section consists of three fields (in this order):

CE_TECH_NAME : Character string

CE_LAYOUT_NAME : Character string

CE_LIB_NAME : Character string

The x-y-t information section lists the number of shape-functions and units for length, width and delay:

CE_NUM_IMPLEMENTATION : Int

CE_LENGTH_UNIT : Character string

CE_TIME_UNIT : Character string

Finally, x-y-t section consists of length-width-delay tuples. All the values in this section are interpreted as floating number.

B.1 An example

The following input lists shape functions for a 4-bit ALU:

```
%COMPONENT INFORMATION
GC_COMPILER_NAME : ALU;
GC_INPUT_WIDTH :     4;
GC_NUM_FUNCTIONS :     2;
GC_FUNCTION_LIST :     GC_ADD,GC_LNOT;
GC_STYLE :     GC_CARRY_LOOKAHEAD;

%TECHNOLOGY INFORMATION
CE_TECH_NAME : FPGA;
CE_LAYOUT_STYLE : SRAM;
CE_LIB_NAME : Xilinx4000;

%X_Y_T INFORMATION
CE_NUM_IMPLEMENTATION : 2;
CE_LENGTH_UNIT : microns;
CE_TIME_UNIT : ns;

%X_Y_T
10 20 30
100.25 200.25 300
```

C Tables

C.1 GENUS_parameterName

The following table lists all of the members of `GENUS_parameterName` and the possible types each can assume. The type is given as a `GENUS_parameterType` symbol.

GENUS Parameter Name	Possible Types
GC_ASYNC_LIST	GC_CharPtrPtrPtr
GC_ATTRIBUTE_PTR	GC_AttributePtr
GC_ATTRIBUTE_FILE	GC_CharPtr
GC_ATTRIBUTE_LIST	GC_AttributeList
GC_CLASS	GC_Class
GC_CLASSES_OP	GC_OpClasses
GC_CLOCK	GC_CharPtr
GC_CLOCK_HIGH	GC_Int
GC_CLOCK_LIST	GC_CharPtrPtrPtr
GC_CLOCK_PERIOD	GC_Int
GC_COMPILER_NAME	GC_CharPtr
GC_COMPONENT_KEY	GC_ComponentKey
GC_COMPONENT_PTR	GC_ComponentPtr
GC_CONSTANT_VALUE	GC_Int
GC_CONTROL_LIST	GC_CharPtrPtrPtr
GC_CONTROL_WIDTH	GC_Int
GC_CORE_FILE	GC_CharPtr
GC_DATA_TYPE	GC_Misc
GC_DELAY_VALUE	GC_Int
GC_DOC_FILE	GC_CharPtr
GC_DRIVE	GC_Misc
GC_DRIVE_LIST	†
GC_DTAS_FILE	GC_CharPtr
GC_ELSE_FLAG	GC_Boolean
GC_ENABLE_FLAG	GC_Boolean
GC_ENABLE_LIST	GC_CharPtr
GC_EXTERNAL_PARAMETERS	GC_ParameterList
GC_FAN_OUT	GC_Int
GC_FILL_INPUT	GC_Int
GC_FUNCTION	GC_Misc
GC_FUNCTION_LIST	GC_OperationList, GC_Operation, GC_CharPtrPtr
GC_GUARD_LIST	GC_EIntPtr
GC_INOUT_LIST	GC_CharPtrPtrPtr
GC_INPUT_LIST	GC_CharPtrPtrPtr
GC_INPUT_WIDTH	GC_Int
GC_INPUT_WIDTH_LIST	GC_IntPtr
GC_INSTANCE_LIST	GC_InstanceList
GC_INSTANCE_NAME	GC_CharPtr
GC_INSTANCE_PTR	GC_InstancePtr

†Not currently defined.

(1 of 3)

GENUS Parameter Name	Possible Types
GC_INTERNAL_PARAMETERS	GC_ParameterList
GC_INVERT_FLAG	GC_Boolean
GC_INVERT_OUT	GC_Boolean
GC_LEFT_INDEX	GC_Int
GC_LEFT_INPUT_WIDTH	GC_Int
GC_LEVEL	GC_Misc
GC_LOGIC_FILE	GC_CharPtr
GC_MAX_PARAMS	GC_Int
GC_MISC_INOUT_LIST	GC_CharPtrPtrPtr
GC_MISC_INPUT_LIST	GC_CharPtrPtrPtr
GC_MISC_OUTPUT_LIST	GC_CharPtrPtrPtr
GC_MODE	GC_Misc
GC_MODE_LIST	GC_MiscList
GC_NUM_ASYNCS	GC_Int
GC_NUM_ATTRIBUTES	GC_Int
GC_NUM_CLOCKS	GC_Int
GC_NUM_CONTROLS	GC_Int
GC_NUM_DRIVES	GC_Int
GC_NUM_ENABLES	GC_Int
GC_NUM_FUNCTIONS	GC_Int
GC_NUM_GUARDS	GC_Int
GC_NUM_INOUT_PORTS	GC_Int
GC_NUM_INOUTS	GC_Int
GC_NUM_INPUTS	GC_Int
GC_NUM_INPUT_PORTS	GC_Int
GC_NUM_LEVELS	GC_Int
GC_NUM_MISC_INOUTS	GC_Int
GC_NUM_MISC_INPUTS	GC_Int
GC_NUM_MISC_OUTPUTS	GC_Int
GC_NUM_MODES	GC_Int
GC_NUM_OUTPUTS	GC_Int
GC_NUM_OUTPUT_PORTS	GC_Int
GC_NUM_PORTS	GC_Int
GC_NUM_STYLES	GC_Int
GC_NUM_TYPES	GC_Int
GC_NUM_WORDS	GC_Int
GC_OP_COMMUTATIVE	GC_Boolean
GC_OP_CONTROL_LIST	GC_CharPtr
GC_OP_INPUT_LIST	GC_CharPtrPtrPtr
GC_OP_NUM_CONTROLS	GC_Int
GC_OP_NUM_FUNCTIONS	GC_Int
GC_OP_NUM_INPUTS	GC_Int
GC_OP_NUM_OUTPUTS	GC_Int
GC_OP_OUTPUT_LIST	GC_CharPtrPtrPtr
GC_OUTPUT_LIST	GC_CharPtrPtrPtr
GC_OUTPUT_WIDTH	GC_Int
GC_OUTPUT_WIDTH_LIST	GC_CharPtrPtrPtr

†Not currently defined.

(2 of 3)

GENUS Parameter Name	**Possible Types**
GC_PARAMETER_LIST	†
GC_PIN_INFO_LIST	GC_ParamPinInfoList
GC_PIPELINE_DELAY	GC_Int
GC_PIPELINE_FLAG	GC_Boolean
GC_PIPELINE_STAGES	GC_Int
GC_PORT_ATTRIBUTE_LIST	†
GC_POS_OUT	†
GC_PROC_INITIALIZE	GC_Proc
GC_PROC_OP_TO_PINS	GC_Proc
GC_PROC_TO_DTAS	GC_Proc
GC_PROC_TO_LOGIC	GC_Proc
GC_PROC_TO_VHDL	GC_Proc
GC_PROC_TO_VHDL_DELAY	GC_Proc
GC_PULSE_WIDTH	GC_Int
GC_RESET_FLAG	GC_Boolean
GC_RIGHT_INDEX	GC_Int
GC_RIGHT_INPUT_WIDTH	GC_Int
GC_SET_FLAG	GC_Boolean
GC_SET_VALUE	GC_Int
GC_SHIFT_DISTANCE	GC_Int
GC_STRENGTH	GC_Int
GC_STYLE	GC_Misc
GC_TYPE	GC_Misc
GC_VHDL_FILE	GC_CharPtr

†Not currently defined.

(3 of 3)

C.2 GENUS_parameterName Operation Parameters

The following table lists all of the distinct operation-specific parameter names. They are also listed in the previous table.

GENUS Parameter Name	Possible Types
GC_OP_COMMUTATIVE	GC_Boolean
GC_OP_CONTROL_LIST	GC_CharPtr
GC_OP_FUNCTION_LIST	†
GC_OP_INPUT_LIST	GC_CharPtrPtrPtr
GC_OP_NUM_CONTROLS	GC_Int
GC_OP_NUM_FUNCTIONS	†
GC_OP_NUM_INPUTS	GC_Int
GC_OP_NUM_OUTPUTS	GC_Int
GC_OP_OUTPUT_LIST	GC_CharPtrPtrPtr

†Not currently defined.

C.3 GENUS_parameterType

The following table lists all of the members of `GENUS_parameterType` and their C type definitions.

GENUS Type Symbol	**C Type Name**
GC_TypeUndef	(*undefined type error*)
GC_Int	int
GC_Misc	GENUS_miscParameterValue
GC_MiscList	GENUS_miscListType * †
GC_EMiscList	GENUS_miscParameterValue *
GC_Class	GENUS_class
GC_Operation	GENUS_operation
GC_OpClasses	GENUS_opClassesListType * †
GC_IntPtr	GENUS_intPtrType * †
GC_EIntPtr	int *
GC_IntPtrPtr	GENUS_intPtrPtrType * ††
GC_EIntPtrPtr	int ** ††
GC_Boolean	Bool
GC_CharPtr	char *
GC_CharPtrPtr	GENUS_charPtrPtrType * †
GC_ECharPtrPtr	char **
GC_CharPtrPtrPtr	GENUS_charPtrPtrPtrType * †
GC_ECharPtrPtrPtr	char ***
GC_ParameterValue	GENUS_parameterValue * ††
GC_ParameterList	GENUS_parameterListType * ††
GC_Proc	(**Boolean)() ††
GC_Generic	caddr_t (*generic pointer*)
GC_InstanceList	GENUS_instanceListType * †
GC_InstancePtr	GENUS_instance *
GC_ComponentKey	GENUS_componentKey †
GC_ComponentPtr	GENUS_component *
GC_OperationList	GENUS_operationListType * †
GC_EOperationList	GENUS_operation * †
GC_AttributeList	GENUS_attributeListType * †‡
GC_AttributePtr	GENUS_attribute * ‡
GC_EAttributeList	GENUS_attribute * †
GC_ShapePtr	GA_SHAPE * †

†Described in Section 8.
††Currently not for external use.
‡Described in Section 7.

C.4 GENUS_class

The following table lists all of the members of `GENUS_class`.

GENUS Class Name
GC_Combinatorial
GC_Clocked
GC_Blackbox

C.5 GENUS_opClass

The following table lists all of the members of `GENUS_opClass`.

GENUS Operation Class Name
GC_OpClassEq
GC_OpClassNEq
GC_OpClassGT
GC_OpClassLT
GC_OpClassGEq
GC_OpClassLEq

C.6 GENUS_miscParameterValue

The following table lists all of the members of `GENUS_miscParameterValue`.

GENUS Misc Parameter Value
GC_ARRAY
GC_BCD
GC_BIDIRECTIONAL
GC_BINARY
GC_BIT
GC_BUFFER
GC_CARRY_LOOKAHEAD
GC_CLOCK_DRIVER
GC_CMOS
GC_CONSTANT
GC_D_FF
GC_ECL
GC_EXTEND
GC_FIFO
GC_FILL
GC_GRAY
GC_HIGH
GC_INPUT
GC_INPUT_OUTPUT
GC_INTEGRATED
GC_ITERATIVE
GC_JOHNSON
GC_LATCH
GC_LOW
GC_MEDIUM
GC_MULTIPLICATIVE
GC_MVL4
GC_NON_RESTORING
GC_ODD
GC_ONE_SHOT
GC_OUTPUT
GC_RESTORING
GC_RIPPLE
GC_RIPPLE_CARRY
GC_STACK
GC_SYNCHRONOUS
GC_TRISTATE
GC_TTL
GC_WALLACE_DADDA

C.7 GENUS_operation

The following table lists all of the members of `GENUS_operation`.

GENUS Operator Name
GC_ADD
GC_AND
GC_BAND
GC_BNAND
GC_BNOR
GC_BNOT
GC_BOR
GC_BXNOR
GC_BXOR
GC_CONTROL_LOGIC
GC_COUNT_DOWN
GC_COUNT_UP
GC_CONCAT
GC_DEC
GC_DECODE
GC_DELAY
GC_DIV
GC_ENCODE
GC_EQ
GC_EXTRACT
GC_GEQ
GC_GT
GC_INC
GC_LEQ
GC_LID
GC_LIMPL
GC_LINHI
GC_LNOT
GC_LOAD
GC_LT
GC_MULT
GC_NAND
GC_NEQ
GC_NOR
GC_ONE
GC_OR
GC_PARITY
GC_POP
GC_PUSH
GC_READ
GC_RID
GC_RIMPL
GC_RINHI
GC_RNOT
GC_ROTL
GC_ROTR

(1 of 2)

GENUS Operator Name
GC_SCAN
GC_SELECT
GC_SHL
GC_SHR
GC_SUB
GC_WIRED_OR
GC_WRITE
GC_XNOR
GC_XOR
GC_ZERO

(2 of 2)

D The GENUS Component Generators

D.1 Format Description

This section describes the GENUS component generators. The listing for each generator is given in the following simplified format description.

```
[generator]_generator(){
   [GENUS_parameterName]:        [value]          ([GENUS_parameterType])
   ...
}

[generator]_component_initialize(){
   [GENUS_parameterName]:        [value]          ([GENUS_parameterType])
   ...
}

/* optional */
_[generator]_control_list([fields]){
   [GENUS_operation] => [GC_CharPtr], ...
}

[generator]_component_op_to_pins([fields]){
   [GENUS_operation], ... :
      [GENUS_parameterName]:     [value]          ([GENUS_parameterType])
      ...
   ...
}

OPERATIONS :
        synchronous behavior
           ...
```

Parts of the `COUNTER` generator will be used as examples.

The first section, `[generator]_generator()`, describes the fixed values for the generator, and the configurable parameters that can be passed in to it.

```
COUNTER_generator(){
   GC_COMPILER_NAME:    COUNTER                 (GC_CharPtr)
   GC_CLASS:            GC_Clocked              (GC_Class)
   GC_NUM_INPUTS:       2                       (GC_Int)
   GC_NUM_OUTPUTS:      2                       (GC_Int)
   GC_CLOCK:            CLK                     (GC_CharPtr)
   GC_NUM_ASYNCS:       2                       (GC_Int)
   GC_ASYNC_LIST:       ASET, ARESET            (GC_CharPtrPtr)
   ...
```

The first entry states that the parameter `GC_COMPILER_NAME` has the value `COUNTER`, which is defined to be type `GC_CharPtr` and is thus of type `char *`. Each successive entry describes the parameter name, the value, and lastly, the type the value has. Multiple values are permitted as is the case for `GC_ASYNC_LIST`. Since the type is defined to be `GC_CharPtrPtr` the list of values will be entered in a list of type `char **`, an array of character strings `"ASET"` and `"ARESET"`.

```
GC_MAX_PARAMS:        11                          (GC_Int)
GC_EXTERNAL_PARAMETERS:      /* variables */
        (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
         GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
         GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                             GC_OperationList,            %fl,
         GC_TYPE,            GC_Required, GC_Misc,        %type,
         GC_STYLE,           GC_Required, GC_Misc,        %style,
         GC_SET_VALUE,       GC_Required, GC_Int,         %setv,
         GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,     %enable
         GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
         GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
         GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                             GC_AttributeList,            %atl,
         GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
```

The `GC_EXTERNAL_PARAMETERS` parameter name denotes a special list of values describing each parameter that can be passed in to the generator. Each entry consists of mainly four elements : name of the parameter, whether it is required or optional, its type and finally a symbol that can be used in the `[generator]_component_initialize()` section. `GC_FUNCTION_LIST` has one extra sub-entry value, `GC_EOperationList`, which describes the type of the argument value passed to **GC_createComponent()** (section 4.3). This type is `GENUS_operation *`, which indicates that `GC_FUNCTION_LIST` will be followed by an array of `GENUS_operation` values in the call. The second sub-entry value, `GC_OperationList`, describes how the list will be represented internally by the component, and also how it will be queried from the component. Thus, a call to **GENUS_query()** (or variation thereof – see Section 5), with `GENUS_FUNCTION_LIST` passed as parameter name will return `GENUS_operationListType *`, which then can be queried with **GENUS_operationListQuery()** (Section 8). See Section 4.1.3 for a discussion of internal versus external parameter types and values. The eighth parameter, namely `GC_DATA_TYPE`, specifies the data-type used for the behavioral models. It is an optional parameter; if left unspecified, model generators use default-value. The last three parameters, namely `GC_NUM_ATTRIBUTES`, `GC_ATTRIBUTE_LIST` and `GC_ATTRIBUTE_FILE`, are also optional parameters. They are used for specifying the performance metrics described in section 7.

```
COUNTER_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)], ISCAN[0]         (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w
   GC_OUTPUT_LIST:      O0[0-(%w-1)], OSCAN[0]         (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _COUNTER_control_list(%f, %fl)  (GC_CharPtrPtr)
}
```

This section describes what values are initialized when a component is realized. In this example, the `GC_INPUT_LIST` value is dependent on the symbol `%w` which was defined in the `COUNTER_generator()` section. The resulting `char *[1][%w]` list (a two-dimensional array of character pointers, the first dimension 1, the second `%w`) will look like:

```
"I0[0]",
"I0[1]",
"I0[2]",
...
```

The `GC_CONTROL_LIST` contains a call to a sub-procedure `_COUNTER_control_list()`. Since the type of the parameter is defined as `GC_CharPtrPtr` this procedure will return an array of character pointers.

```
_COUNTER_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_LOAD => "CLOAD", GC_COUNT_UP => "CUP", GC_COUNT_DOWN => "CDOWN",
   GC_SCAN => "CSCAN"
}
```

This procedure does not explicitly return an array since all that is really needed to be shown is the relationship between operator names and control names. Once called, the `GC_CONTROL_LIST` parameter will contain a list of strings corresponding one-to-one with the operator names in the function list.

```
COUNTER_component_op_to_pins((GC_Operation)%o){

  GC_LOAD:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS          (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST[0]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       1                      (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST[0]      (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:      1                      (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]    (GC_CharPtr)
   ...
```

This section of a generator listing shows the relationship of input, output, and control pins, to operators. In this example, `GC_LOAD` results in each operation-specific parameter (see section 5.4) being assigned the specified value. For example, `GC_OP_NUM_INPUTS` and `GC_OP_NUM_OUTPUTS` are assigned 1. `GC_OP_CONTROL_LIST` is assigned an indexed value of the `GC_CONTROL_LIST`. This is straightforward since operators and control pin names were assigned one-to-one.

```
OPERATIONS :
     Synchronous behavior
        LOAD : O0 = I0;
        COUNT_UP : O0 = O0 + 1;
        COUNT_DOWN : O0 = O0 - 1;
        SCAN : O0 = ISCAN & O0(%w-1,1); OSCAN = O0(0);
    Asynchronous behavior
        SET : O0 = %setv;
        RESET : O0 = 0;
```

The last section of generator listing informally describes the functionality and idiosyncrasies of the components. The behavioral models described in the section 9 mimic the operations mentioned in this section.

D.2 ADDSUB

```
ADD_SUB_generator(){
   GC_COMPILER_NAME:    ADD_SUB                 (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       3                       (GC_Int)
   GC_NUM_OUTPUTS:      3                       (GC_Int)
   GC_NUM_CONTROLS:     3                       (GC_Int)
   GC_CONTROL_LIST:     CADD, CSUB, CRSUB       (GC_CharPtrPtr)
   GC_NUM_FUNCTIONS:    3                       (GC_Int)
   GC_FUNCTION_LIST:    GC_ADD, GC_SUB, GC_RSUB (GC_OperationList)
   GC_NUM_STYLES:       4                       (GC_Int)
   GC_STYLE_LIST:       GC_RIPPLE_CARRY,
                        GC_CARRY_LOOKAHEAD,
                        GC_MEDIUM,
                        GC_CARRY_SAVE           (GC_MiscList)

   GC_MAX_PARAMS:       8                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl,
            GC_STYLE,           GC_Required, GC_Misc,        %s,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

ADD_SUB_component_initialize(){
   GC_INPUT_LIST:       I0-1[0-(%w-1)], ICIN[0]         (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                              (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)], OCOUT[0], OVF[0]  (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _ADD_SUB_control_list(%f, %fl)  (GC_CharPtrPtr)
}

_ADD_SUB_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_ADD  => "CADD"
   GC_SUB  => "CSUB"
   GC_RSUB => "CRSUB"
}

ADD_SUB_component_op_to_pins((GC_Operation)%o){
   GC_ADD:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_SUB,GC_RSUB:
      GC_OP_COMMUTATIVE:        False                   (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
```

```
    GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
    GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST         (GC_CharPtrPtrPtr)
    GC_OP_NUM_CONTROLS:      1                      (GC_Int)
    GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]    (GC_CharPtr)
}

OPERATIONS :
        ADD : O0 = I0 + I1;
        SUB : O0 = I0 - I1;
        RSUB : O0 = I1 - I0;

        Inputs and Outputs are in two's complement form.
```

D.3 ALU

```
ALU_generator(){
   GC_COMPILER_NAME:    ALU                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       3                       (GC_Int)
   GC_NUM_OUTPUTS:      5                       (GC_Int)
   GC_NUM_CONTROLS:     27                      (GC_Int)
   GC_CONTROL_LIST:     CADD, CSUB, CRSUB, CINC, CDEC, CEQ, CNEQ, CGT, CLT,
                        CGEQ, CLEQ, CZERO, CONE, CAND, CNAND, COR, CNOR,
                        CXOR, CXNOR, CLID, CRID, CLNOT, CRNOT, CRINHI,
                        CLINHI, CRIMPL, CLIMPL  (GC_CharPtrPtr)
   GC_NUM_FUNCTIONS:    27                      (GC_Int)
   GC_FUNCTION_LIST:    GC_ADD, GC_SUB, GC_RSUB, GC_INC, GC_DEC, GC_EQ,
                        GC_NEQ, GC_GT, GC_LT, GC_GEQ, GC_LEQ, GC_ZERO,
                        GC_ONE, GC_AND, GC_NAND, GC_OR, GC_NOR, GC_XOR,
                        GC_XNOR, GC_LID, GC_RID, GC_LNOT, GC_RNOT, GC_RINHI,
                        GC_LINHI, GC_RIMPL, GC_LIMPL
                                                (GC_OperationList)
   GC_NUM_STYLES:       3                       (GC_Int)
   GC_STYLE:            GC_RIPPLE_CARRY,
                        GC_CARRY_LOOKAHEAD,
                        GC_INTEGRATED           (GC_MiscList)

   GC_MAX_PARAMS:       10                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
          (GC_INPUT_WIDTH,      GC_Required, GC_Int,         %w,
           GC_NUM_FUNCTIONS,    GC_Required, GC_Int,         %f,
           GC_FUNCTION_LIST,    GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl,
           GC_STYLE,            GC_Required, GC_Misc,        %s,
           GC_PIPELINE_FLAG,    GC_Optional, GC_Boolean,     %pf,
           GC_PIPELINE_STAGES,  GC_Optional, GC_Int,         %ps,
           GC_DATA_TYPE,        GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,   GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,   GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,   GC_Optional, GC_CharPtr      %af);
}

ALU_component_initialize(){
   GC_INPUT_LIST:       I0-1[0-(%w-1)], ICIN[0]         (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                              (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)], O1[0], OCOUT[0],
                        OZERO[0], OVF[0]                (GC_CharPtrPtrPtr)
```

```
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _ALU_control_list(%f, %fl)      (GC_CharPtrPtr)
}

_ALU_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_ADD => "CADD", GC_SUB => "CSUB", GC_RSUB => "CSUB", GC_INC => "CINC",
   GC_DEC => "CDEC", GC_EQ => "CEQ", GC_NEQ => "CNEQ", GC_GT => "CGT",
   GC_LT => "CLT", GC_GEQ => "CGEQ", GC_LEQ => "CLEQ", GC_ZERO => "CZERO",
   GC_ONE => "CONE", GC_AND => "CAND", GC_NAND => "CNAND", GC_OR => "COR",
   GC_NOR => "CNOR", GC_XOR => "CXOR", GC_XNOR => "CXNOR", GC_LID => "CLID",
   GC_RID => "CRID", GC_LNOT => "CLNOT", GC_RNOT => "CRNOT",
   GC_RINHI => "CRINHI", GC_LINHI => "CLINHI", GC_RIMPL => "CRIMPL",
   GC_LIMPL => "CLIMPL
}

ALU_component_op_to_pins((GC_Operation)%o){

   GC_ADD:
      GC_OP_COMMUTATIVE:        True                     (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS            (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST            (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        3                        (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,2,3,4] (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                        (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_SUB,GC_RSUB:
      GC_OP_COMMUTATIVE:        False                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS            (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST            (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        3                        (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,2,3,4] (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                        (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_INT, GC_DEC:
      GC_OP_COMMUTATIVE:        False                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         2                        (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0,2]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        4                        (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,2,3,4] (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                        (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_EQ, GC_NEQ:
      GC_OP_COMMUTATIVE:        True                     (GC_Boolean)
      GC_OP_NUM_INPUTS:         2                        (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0,1]       (GC_Int)
      GC_OP_NUM_OUTPUTS:        1                        (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[1]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                        (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_GT, GC_LT, GC_GEQ, GC_LEQ:
      GC_OP_COMMUTATIVE:        False                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         2                        (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0,1]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                        (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[1]        (GC_CharPtrPtrPtr)
```

```
      GC_OP_NUM_CONTROLS:       1                      (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]    (GC_CharPtr)

   GC_ZERO, GC_ONE:
      GC_OP_COMMUTATIVE:        True                   (GC_Boolean)
      GC_OP_NUM_INPUTS:         0                      (GC_Int)
      GC_OP_NUM_OUTPUTS:        2                      (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,3]    (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                      (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]    (GC_CharPtr)

   GC_AND, GC_NAND, GC_OR, GC_NOR, GC_XOR, GC_XNOR:
      GC_OP_COMMUTATIVE:        True                   (GC_Boolean)
      GC_OP_NUM_INPUTS:         2                      (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0,1]     (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        2                      (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,3]    (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                      (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]    (GC_CharPtr)

   GC_LID, GC_LNOT:
      GC_OP_COMMUTATIVE:        False                  (GC_Boolean)
      GC_OP_NUM_INPUTS:         1                      (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        2                      (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,3]    (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                      (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]    (GC_CharPtr)

   GC_RID, GC_RNOT:
      GC_OP_COMMUTATIVE:        False                  (GC_Boolean)
      GC_OP_NUM_INPUTS:         1                      (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[1]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        2                      (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,3]    (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                      (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]    (GC_CharPtr)

   GC_RINHI, GC_LINHI, RIMPL, LIMPL:
      GC_OP_COMMUTATIVE:        False                  (GC_Boolean)
      GC_OP_NUM_INPUTS:         2                      (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0,1]     (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        2                      (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,3]    (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                      (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]    (GC_CharPtr)
}

OPERATIONS :
        ADD : O0 = I0 + I1;
        SUB : O0 = I0 - I1;
        RSUB : O0 = I1 - I0;
        INC : O0 = I0 + 1;
        DEC : O0 = I0 - 1;
        EQ : if (I0 = I1) then O1 = 1; else O1 = 0;
        NEQ : if (I0 <> I1) then O1 = 1; else O1 = 0;
        GT : if (I0 > I1) then O1 = 1; else O1 = 0;
        LT : if (I0 < I1) then O1 = 1; else O1 = 0;
        GEQ : if (I0 >= I1) then O1 = 1; else O1 = 0;
```

```
LEQ : if (I0 <= I1) then O1 = 1; else O1 = 0;
ZERO : O0 = '0..0';
ONE : O0 = '1...1';
AND : O0 = I0 and I1;
NAND : O0 = I0 and I1;
OR : O0 = I0 or I1;
NOR : O0 = I0 nor I1;
XOR : O0 = I0 xor I1;
XNOR : O0 = I0 xnor I1;
LID : O0 = I0;
RID : O0 = I1;
LNOT : O0 = not (I0);
RNOT : O0 = not (I1);
RINHI : O0 = I0 and not (I1);
LINHI : O0 = not (I0) and I1;
RIMPL : O0 = I0 or not (I1);
LIMPL : O0 = not(I0) or I1;

if (O0 = 0) then OZERO = 1; else OZERO = 0;
```

D.4 AND

```
AND_generator(){
   GC_COMPILER_NAME:    AND                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_AND                  (GC_Operation)
   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:     /* variables */
          (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
           GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
           GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                               GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

AND_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

AND_component_op_to_pins((GENUS_operation) %o){
   GC_AND:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 and I1 ... and I(n-1);
```

D.5 BARRELSHIFTER

```
BARREL_SHIFTER_generator(){

   GC_COMPILER_NAME:    BARREL_SHIFTER          (CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       3                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_CONTROLS:     4                       (GC_Int)
   GC_CONTROL_LIST:     CSHL, CSHR, CROTR, CROTL
                                                (GC_CharPtrPtr)
   GC_NUM_FUNCTIONS:    4                       (GC_Int)
   GC_FUNCTION_LIST:    GC_SHL, GC_SHR, GC_ROTR, GC_ROTL
                                                (GC_OperationList)

   GC_MAX_PARAMS:       8                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl,
            GC_SHIFT_DISTANCE,  GC_Required, GC_Int,         %s,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

BARREL_SHIFTER_component_initialize(){

   GC_INPUT_LIST:       I0[0-(%w-1)], ISHNUM[0-ln(%w)],
                        IFILL[0]                        (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                              (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]                    (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _BARREL_SHIFTER_control_list(%f, %fl)   (GC_CharPtrPtr)
}

_BARREL_SHIFTER_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_SHL => "CSHL", GC_SHR => "CSHR", GC_ROTR => "CROTR",
   GC_ROTL => "CROTL", GC_ASHL => "CASHL", GC_ASHR => "CASHR"
}

BARREL_SHIFTER_component_op_to_pins((GC_Operation)%o){
   GC_SHL, GC_SHR, GC_ROTR, GC_ROTL:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)
}

OPERATIONS :
        SHL : Shift left by ISHNUM bits using IFILL to fill the right bits;
        SHR : Shift right by ISHNUM bits using IFILL to fill the left bits;
        ROTL : Rotate left by ISHNUM bits;
        ROTR : Rotate right by ISHNUM bits;
```

D.6 BUFFER

```
BUFFER_generator(){
   GC_COMPILER_NAME:    BUFFER                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_BUFFER               (GC_Misc)
   GC_MAX_PARAMS:       7                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_STRENGTH,        GC_Optional, GC_int,         %s,
            GC_INVERT_FLAG,     GC_Optional, GC_Boolean,     %io,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

BUFFER_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

BUFFER_component_op_to_pins((GENUS_misc) %o){
   GC_BUFFER:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        if (%io) then O0 = not I0; else O0 = I0;
```

D.7 BUS

```
BUS_generator(){
   GC_COMPILER_NAME:    BUS                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_MAX_PARAMS:       7                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
            GC_FAN_OUT,         GC_Required, GC_Int,         %fo,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

BUS_component_initialize(){
```

```
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]      (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]             (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    %n                       (GC_Int)
   GC_FUNCTION_LIST:    I0-(%n-1)                (GC_CharPtrPtr)
   GC_NUM_CONTROLS:     %w                       (GC_Int)
   GC_CONTROL_LIST:     CI0-(%n-1)               (GC_CharPtrPtr)
}

BUS_component_op_to_pins((GC_CharPtr) %o){
   %o:
      GC_OP_NUM_INPUTS:        1                       (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST[%o]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       1                       (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:      1                       (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%O]     (GC_CharPtrPtr)
}

OPERATIONS :
        The input, whose control input is 1, is selected. Behaviour wise,
        there is no difference between a MUX and a BUS.
```

D.8 CLOCKGEN

```
CLOCK_GEN_generator(){
   GC_COMPILER_NAME:    CLOCK_GEN                (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial         (GC_Class)
   GC_NUM_INPUTS:       0                        (GC_Int)
   GC_NUM_OUTPUTS:      1                        (GC_Int)
   GC_OUTPUT_WIDTH:     1                        (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                    (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                        (GC_Int)
   GC_FUNCTION_LIST:    GC_CLOCK_GEN             (GC_Misc)

   GC_MAX_PARAMS:       6                        (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_CLOCK_PERIOD,    GC_Required, GC_Int,         %p,
            GC_CLOCK_HIGH,      GC_Required, GC_Int,         %h,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

CLOCK_GEN_component_op_to_pins((GENUS_misc) %o){
   GC_CLOCK_GEN:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = Clock with period %p and duration for it remains high is %h.
```

D.9 COMPARATOR

```
COMPARATOR_generator(){
   GC_COMPILER_NAME:    COMPARATOR              (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       2                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)

   GC_NUM_FUNCTIONS:    6                       (GC_Int)
   GC_FUNCTION_LIST:    GC_EQ, GC_NEQ, GC_GT, GC_LT, GC_GEQ, GC_LEQ
                                                (GC_OperationList)

   GC_MAX_PARAMS:       7                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

COMPARATOR_component_initialize(){
   GC_INPUT_LIST:       I0-1[0-(%w-1)]                  (GC_CharPtrPtrPtr)
   GC_NUM_OUTPUTS:      %f
   GC_OUTPUT_LIST:      _COMPARATOR_output_list(%f, %fl)        (GC_CharPtrPtr)
}

_COMPARATOR_output_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_EQ => "OEQ[0]", GC_NEQ => "ONEQ[0]", GC_GT => "OGT[0]",
   GC_LT => "OLT[0]", GC_GEQ => "OGEQ[0]", GC_LEQ => "OLEQ[0]"
}

COMPARATOR_component_op_to_pins((GC_Operation)%o){

   GC_EQ, GC_NEQ :
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%o]      (GC_CharPtrPtrPtr)

   GC_GT, GC_LT, GC_GEQ, GC_LEQ :
      GC_OP_COMMUTATIVE:        False                   (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%o]      (GC_CharPtrPtrPtr)
}

OPERATIONS :
        if (I0 = I1) then OEQ = 1; else OEQ = 0;
        if (I0 <> I1) then ONEQ = 1; else ONEQ = 0;
        if (I0 < I1) then OLT = 1; else OLT = 0;
        if (I0 > I1) then OGT = 1; else OGT = 0;
        if (I0 <= I1) then OLEQ = 1; else OGEQ = 0;
```

```
        if (I0 >= I1) then OGEQ = 1; else OGEQ = 0;
```

D.10 CONCAT

```
CONCAT_generator(){
   GC_COMPILER_NAME:    CONCAT                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_CONCAT               (GC_Operation)

   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
            GC_INPUT_WIDTH_LIST,GC_Required, GC_EIntPtr,
                                GC_IntPtr,                   %il,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

CONCAT_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-{%il(j)}]   (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     sum(%il{j})             (GC_Int)
   GC_OUTPUT_LIST:      O0[0-sum(%il{j})]       (GC_CharPtrPtrPtr)
}

CONCAT_component_op_to_pins((GENUS_operation) %o){
   GC_CONCAT:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 & I1 ... & I(n-1);
```

D.11 CONSTANT

```
CONSTANT_generator(){
   GC_COMPILER_NAME:    CONSTANT                (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       0                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_CONSTANT             (GC_Misc)
   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_OUTPUT_WIDTH,    GC_Required, GC_Int,         %w,
            GC_CONST_VALUE,     GC_Required, GC_Int,         %v,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
```

```
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

CONSTANT_component_initialize(){
   GC_OUTPUT_LIST:      O0[0-(%w-1)]           (GC_CharPtrPtrPtr)
}

CONSTANT_component_op_to_pins((GENUS_misc) %o){
   GC_CONSTANT:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS          (GC_Int)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST         (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = %v;
```

D.12 COUNTER

```
COUNTER_generator(){
   GC_COMPILER_NAME:    COUNTER                (GC_CharPtr)
   GC_CLASS:            GC_Clocked             (GC_Class)
   GC_NUM_INPUTS:       2                      (GC_Int)
   GC_NUM_OUTPUTS:      2                      (GC_Int)
   GC_CLOCK:            CLK                    (GC_CharPtr)
   GC_NUM_ASYNCS:       2                      (GC_Int)
   GC_ASYNC_LIST:       ASET, ARESET           (GC_CharPtrPtr)

   GC_NUM_CONTROLS:     4                      (GC_Int)
   GC_CONTROL_LIST:     CLOAD,CUP,CDOWN,CSCAN  (GC_CharPtrPtr)

   GC_NUM_FUNCTIONS:    4                      (GC_Int)
   GC_FUNCTION_LIST:    GC_LOAD, GC_COUNT_UP, GC_COUNT_DOWN,
                        GC_SCAN,               (GC_OperationList)

   GC_NUM_TYPES:        4                      (GC_Int)
   GC_TYPE_LIST:        GC_BINARY, GC_BCD, GC_JOHNSON, GC_GRAY
                                               (GC_MiscList)

   GC_NUM_STYLES:       2                      (GC_Int)
   GC_STYLE_LIST:       GC_SYNCHRONOUS, GC_RIPPLE
                                               (GC_MiscList)

   /* optional */
   GC_NUM_ENABLES:      1
   GC_ENABLE_LIST:      CEN

   GC_MAX_PARAMS:       11                      (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl,
            GC_TYPE,            GC_Required, GC_Misc,        %type,
            GC_STYLE,           GC_Required, GC_Misc,        %style,
            GC_SET_VALUE,       GC_Required, GC_Int,         %setv,
            GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,     %enable
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
```

```
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

COUNTER_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)], ISCAN[0]          (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w
   GC_OUTPUT_LIST:      O0[0-(%w-1)], OSCAN[0]          (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _COUNTER_control_list(%f, %fl)  (GC_CharPtrPtr)
}

_COUNTER_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_LOAD => "CLOAD", GC_COUNT_UP => "CUP", GC_COUNT_DOWN => "CDOWN",
   GC_SCAN => "CSCAN"
}

COUNTER_component_op_to_pins((GC_Operation)%o){

   GC_LOAD:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST[0]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       1                       (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST[0]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:      1                       (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_COUNT_UP:
   GC_COUNT_DOWN:
      GC_OP_NUM_INPUTS:        0                       (GC_Int)
      GC_OP_NUM_OUTPUTS:       1                       (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST[0]          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:      1                       (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_SCAN:
      GC_OP_NUM_INPUTS:        1                       (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST[1]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       1                       (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:      1                       (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]     (GC_CharPtr)
}

OPERATIONS :
     Synchronous behaviour
        LOAD : O0 = I0;
        COUNT_UP : O0 = O0 + 1;
        COUNT_DOWN : O0 = O0 - 1;
        SCAN : O0 = ISCAN & O0(%w-1,1); SCAN = O0(0);
    Asynchronous behaviour
        SET : O0 = %setv;
        RESET : O0 = 0;
```

D.13 DECODE

```
DECODE_generator(){
   GC_COMPILER_NAME:    DECODE                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_DECODE               (GC_Operation)

   GC_NUM_TYPES:        2                       (GC_Int)
   GC_TYPE_LIST:        GC_BINARY, GC_BCD       (GC_MiscList)

   GC_MAX_PARAMS:       5                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

DECODE_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_OUTPUT_LIST:      O0[0-exp(%w)]           (GC_CharPtrPtrPtr)
}

DECODE_component_op_to_pins((GENUS_operation) %o){
   GC_DECODE:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = decode(I0);
```

D.14 DELAY

```
DELAY_generator(){
   GC_COMPILER_NAME:    DELAY                   (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUT:        1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_DELAY                (GC_Operation)

   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_DELAY_VALUE,     GC_Required, GC_Int,         %d,
            GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
```

```
           GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

DELAY_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

DELAY_component_op_to_pins((GENUS_operation) %o){
   GC_DELAY:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS          (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST         (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 with DELAY %d ns;
```

D.15 DIV

```
DIV_generator(){
   GC_COMPILER_NAME:    DIV                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       2                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_DIV                  (GC_Operation)

   GC_NUM_STYLES:       3                       (GC_Int)
   GC_STYLE_LIST:       GC_RESTORING, GC_NON_RESTORING, GC_MULTIPLICATIVE
                                                (GC_MiscList)

   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:     /* variables */
          (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
           GC_STYLE,           GC_Required, GC_Misc,        %s,
           GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                               GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

DIV_component_initialize(){
   GC_INPUT_LIST:       I0-1[0-(%w-1)]          (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

DIV_component_op_to_pins((GENUS_operation) %o){
   GC_DIV:
      GC_OP_COMMUTATIVE:       False                  (GC_Boolean)
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS          (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
```

```
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 div I1;
```

D.16 ENCODE

```
ENCODE_generator(){
   GC_COMPILER_NAME:    ENCODE                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_ENCODE               (GC_Operation)

   GC_MAX_PARAMS:       5                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %n,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

ENCODE_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_OUTPUT_LIST:      O0[0-(ln(%w)-1)]        (GC_CharPtrPtrPtr)
}

ENCODE_component_op_to_pins((GENUS_operation) %o){
   GC_ENCODE:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = encode(I0);
```

D.17 EXTRACT

```
EXTRACT_generator(){
   GC_COMPILER_NAME:    EXTRACT                 (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_EXTRACT              (GC_Operation)

   GC_MAX_PARAMS:       7                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_LEFT_INDEX,      GC_Required, GC_Int,         %l,
```

```
            GC_RIGHT_INDEX,     GC_Required, GC_Int,        %r,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

EXTRACT_component_initialize(){
   GC_INPUT_LIST:      I0[0-(%w-1)]           (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:    %l-%r+1                (GC_Int)
   GC_OUTPUT_LIST:     O0[0-(%l-%r)]          (GC_CharPtrPtrPtr)
}

EXTRACT_component_op_to_pins((GENUS_operation) %o){
   GC_EXTRACT:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS          (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST         (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0(%l..%r);
        Assumption : Left most bit is the most significant bit.
```

D.18 FIFO

```
FIFO_generator(){
   GC_COMPILER_NAME:   FIFO                   (GC_CharPtr)
   GC_CLASS:           GC_Clocked             (GC_Class)
   GC_NUM_INPUTS:      1                      (GC_Int)
   GC_NUM_OUTPUTS:     1                      (GC_Int)

   GC_NUM_CONTROLS:    2                      (GC_Int)
   GC_CONTROL_LIST     CPUSH, CPOP            (GC_CharPtrPtr)

   GC_CLOCK            CLK                    (GC_CharPtr)

   GC_NUM_ENABLES:     1                      (GC_Int)
   GC_ENABLE_LIST:     CEN                    (GC_CharPtr)

   GC_NUM_FUNCTIONS:   2                      (GC_Int)
   GC_FUNCTION_LIST:   GC_PUSH, GC_POP        (GC_OperationList)

   GC_MAX_PARAMS:      7                      (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,        %w,
            GC_NUM_WORDS,       GC_Required, GC_Int,        %nw,
            GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,    %e,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

FIFO_component_initialize(){
```

```
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_CONTROL_LIST:     CPUSH, CPOP             (GC_CharPtrPtr)
}

FIFO_component_op_to_pins((GENUS_operation) %o){
   GC_PUSH, GC_POP:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS          (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST         (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:      1                      (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]    (GC_CharPtr)
}

OPERATIONS :
     Synchronous behaviour
        PUSH : Storage[head] = I0; head = head + 1;
        POP : O0 = Storage[tail]; tail = tail + 1;
     Assumption : No Overflow or Underflow
```

D.19 GAND

```
GAND_generator(){
   GC_COMPILER_NAME:    GAND                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_BAND                 (GC_Operation)
   GC_MAX_PARAMS:       5                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,        %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

GAND_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

GAND_component_op_to_pins((GC_Operation)%o){
   GC_BAND:
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0[0] and I0[1] ... and I0[%w-1];
```

D.20 GNAND

```
GNAND_generator(){
   GC_COMPILER_NAME:    GNAND                   (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_BNAND                (GC_Operation)
   GC_MAX_PARAMS:       5                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

GNAND_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

GNAND_component_op_to_pins((GC_Operation)%o){
   GC_BNAND:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = not (I0[0] and I0[1] ... and I0[%w-1]);
```

D.21 GNOR

```
GNOR_generator(){
   GC_COMPILER_NAME:    GNOR                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_BNOR                 (GC_Operation)
   GC_MAX_PARAMS:       5                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

GNOR_component_initialize(){
```

```
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

GNOR_component_op_to_pins((GC_Operation)%o){
   GC_BNOR:
      GC_OP_NUM_INPUTS:          GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = not(I0[0] or I0[1] ... or I0[%w-1]);
```

D.22 GNOT

```
GNOT_generator(){
   GC_COMPILER_NAME:    GNOT                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_INPUT_WIDTH       1                       (GC_Int)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_INPUT_LIST:       I0[0]                   (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_BNOT                 (GC_Operation)
   GC_MAX_PARAMS:       4                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
           (GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

GNOT_component_op_to_pins((GC_Operation)%o){
   GC_BNOT:
      GC_OP_NUM_INPUTS:          GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = not (I0);
```

D.23 GOR

```
GOR_generator(){
   GC_COMPILER_NAME:    GOR                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
```

```
   GC_FUNCTION_LIST:    GC_BOR                 (GC_Operation)
   GC_MAX_PARAMS:       5                      (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
          (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
           GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                               GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

GOR_component_initialize(){
   GC_INPUT_LIST:       IO[0-(%w-1)]           (GC_CharPtrPtrPtr)
}

GOR_component_op_to_pins((GC_Operation)%o){
   GC_BOR:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
        OO = IO[0] OR IO[1] ... OR IO[%w-1];
```

D.24 GXNOR

```
GXNOR_generator(){
   GC_COMPILER_NAME:    GXNOR                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial       (GC_Class)
   GC_NUM_INPUTS:       1                      (GC_Int)
   GC_OUTPUT_WIDTH:     1                      (GC_Int)
   GC_NUM_OUTPUTS:      1                      (GC_Int)
   GC_OUTPUT_LIST:      OO[0]                  (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                      (GC_Int)
   GC_FUNCTION_LIST:    GC_BXNOR               (GC_Operation)
   GC_MAX_PARAMS:       5                      (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
          (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
           GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                               GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

GXNOR_component_initialize(){
   GC_INPUT_LIST:       IO[0-(%w-1)]           (GC_CharPtrPtrPtr)
}

GXNOR_component_op_to_pins((GC_Operation)%o){
   GC_BXNOR:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}
```

```
OPERATIONS :
        O0 = not(I0[0] xor I0[1] ... xor I0[%w-1]);
```

D.25 GXOR

```
GXOR_generator(){
   GC_COMPILER_NAME:    GXOR                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_BXOR                 (GC_Operation)
   GC_MAX_PARAMS:       5                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

GXOR_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

GXOR_component_op_to_pins((GC_Operation)%o){
   GC_BXOR:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0[0] xor I0[1] ... xor I0[%w-1];
```

D.26 LU

```
LU_generator(){
   GC_COMPILER_NAME:    LU                      (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       2                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_CONTROLS:     16                      (GC_Int)
   GC_CONTROL_LIST:     CZERO, CONE, CAND, CNAND, COR, CNOR, CXOR, CXNOR
                        CLID, CRID, CLNOT, CRNOT, CRINHI, CLINHI, CRIMPL,
                        CLIMPL                  (GC_CharPtrPtr)
   GC_NUM_FUNCTIONS:    16                      (GC_Int)
   GC_FUNCTION_LIST:    GC_ZERO, GC_ONE, GC_AND, GC_NAND, GC_OR, GC_NOR,
                        GC_XOR, GC_XNOR, GC_LID, GC_RID, GC_LNOT, GC_RNOT,
                        GC_RINHI, GC_LINHI, GC_RIMPL, GC_LIMPL
                                                (GC_OperationList)
```

```
   GC_MAX_PARAMS:       7                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

LU_component_initialize(){
   GC_INPUT_LIST:       I0-1[0-(%w-1)]                (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                            (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]                  (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _LU_control_list(%f, %fl)     (GC_CharPtrPtr)
}

_LU_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_ZERO => "CZERO", GC_ONE => "CONE", GC_AND => "CAND",
   GC_NAND => "CNAND", GC_OR => "COR", GC_NOR => "CNOR",
   GC_XOR => "CXOR", GC_XNOR => "CXNOR", GC_LID => "CLID", GC_RID => "CRID",
   GC_LNOT => "CLNOT", GC_RNOT => "CRNOT", GC_RINHI => "CRINHI",
   GC_LINHI => "CLINHI", GC_RIMPL => "CRIMPL", GC_LIMPL => "CLIMPL"
}

LU_component_op_to_pins((GC_Operation)%o){

   GC_AND, GC_NAND, GC_OR, GC_NOR, GC_XOR, GC_XNOR:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_ZERO, GC_ONE:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         0                       (GC_Int)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_LID, GC_LNOT:
      GC_OP_COMMUTATIVE:        False                   (GC_Boolean)
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_RID, GC_RNOT:
      GC_OP_COMMUTATIVE:        False                   (GC_Boolean)
```

```
    GC_OP_NUM_INPUTS:         1                     (GC_Int)
    GC_OP_INPUT_LIST:         GC_INPUT_LIST[1]      (GC_CharPtrPtrPtr)
    GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS        (GC_Int)
    GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST        (GC_CharPtrPtrPtr)
    GC_OP_NUM_CONTROLS:       1                     (GC_Int)
    GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]   (GC_CharPtr)

  GC_LINHL, GC_LIMPL, GC_RINHI, GC_RIMPL:
    GC_OP_COMMUTATIVE:        False                 (GC_Boolean)
    GC_OP_NUM_INPUTS:         GC_NUM_INPUT          (GC_Int)
    GC_OP_INPUT_LIST:         GC_INPUT_LIST         (GC_CharPtrPtrPtr)
    GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS        (GC_Int)
    GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST        (GC_CharPtrPtrPtr)
    GC_OP_NUM_CONTROLS:       1                     (GC_Int)
    GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]   (GC_CharPtr)
}

OPERATIONS :
        ZERO : O0 = '0..0';
        ONE : O0 = '1...1';
        AND : O0 = I0 and I1;
        NAND : O0 = I0 and I1;
        OR : O0 = I0 or I1;
        NOR : O0 = I0 nor I1;
        XOR : O0 = I0 xor I1;
        XNOR : O0 = I0 xnor I1;
        LID : O0 = I0;
        RID : O0 = I1;
        LNOT : O0 = not (I0);
        RNOT : O0 = not (I1);
        RINHI : O0 = I0 and not (I1);
        LINHI : O0 = not (I0) and I1;
        RIMPL : O0 = I0 or not(I1);
        LIMPL : O0 = not(I0) or I1;
```

D.27 MEMORY

```
MEMORY_generator(){
   GC_COMPILER_NAME:    MEMORY                  (GC_CharPtr)
   GC_CLASS:            GC_Clocked              (GC_Class)
   GC_CLOCK:            CLK                     (GC_CharPtr)
   GC_NUM_ENABLES:      1                       (GC_Int)
   GC_ENABLE_LIST:      CEN                     (GC_CharPtr)

   GC_MAX_PARAMS:       10                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
          (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %iw,
           GC_NUM_WORDS,       GC_Required, GC_Int,         %nw,
           GC_NUM_INPUT_PORTS, GC_Required, GC_Int,         %ni,
           GC_NUM_OUTPUT_PORTS,GC_Required, GC_Int,         %no,
           GC_NUM_INOUT_PORTS, GC_Required, GC_Int,         %nio,
           GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,     %e,
           GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                               GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}
```

```
MEMORY_component_initialize(){
   GC_NUM_INPUTS:       2 * %ni + 2 * %nio + %no    (GC_Int)
   GC_INPUT_LIST:       I0-(%ni-1)[0-(%iw-1)],
                        IO0-(%nio-1)[0-(%iw-1)],
                        IW0-(%ni-1)[0-(ln(%iw)-1)],
                        IR0-(%no-1)[0-(ln(%iw)-1)],
                        IW0-(%nio-1)[0-(ln(%iw)-1)] (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                          (GC_Int)
   GC_NUM_OUTPUTS:      %no + %nio                  (GC_Int)
   GC_OUTPUT_LIST:      O0-(%no-1)[0-(%iw-1)],
                        IO0-(%nio-1)[0-(%iw-1)]     (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    %ni + %no + %nio
   GC_FUNCTION_LIST:    W0-(%ni-1), R0-(%no-1),
                        RW0-(%nio-1)                (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %ni + %no + %nio            (GC_Int)
   GC_CONTROL_LIST:     CW0-(%ni-1), CR0-(%no-1),
                        CRW0-1[0-(%nio-1)]          (GC_CharPtrPtrPtr)
}

MEMORY_component_op_to_pins((GC_CharPtr) %o){
   read_op(%o):
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[2*%ni+%nio+%o]
                                                        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%o]      (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%ni+%o] (GC_CharPtrPtrPtr)

   write_op(%o):
      GC_OP_NUM_INPUTS:         2                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[%o, %ni+%nio+%o]
                                                        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        0                       (GC_Int)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtrPtrPtr)

   read_write_op(%o):
      GC_OP_NUM_INPUTS:         2                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[%ni+%o, 2*%ni+%nio+%no+%o]
                                                        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%no+%o]  (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%ni+%no+%o]
                                                        (GC_CharPtrPtrPtr)
}
OPERATIONS :
   WRITE :
      if (CW0 == 1) then storage[IW0] = I0;
      if (CW1 == 1) then storage[IW1] = I1;
                    :
                    :
                    :
      if (CW(%ni-1) == 1) then storage[IW(%ni-1)] = I(%ni-1);
   READ :
      if (CR0 == 1) then O0 = storage[IR0];
      if (CR1 == 1) then O1 = storage[IR1];
```

```
                     :
                     :
                     :
      if (CR(%no-1) == 1) then storage[IR(%no-1)] = I(%no-1);
   READ_WRITE :
      if (CRW0(0) == 1) then
         if (CRW0(1) == 1) then IO0 = storage[IRW0]
                           else storage[IRW0] = IO0;
      if (CRW1(0) == 1) then
         if (CRW1(1) == 1) then IO1 = storage[IRW1]
                           else storage[IRW1] = IO1;
                     :
                     :
                     :
      if (CRW(%nio-1)(0) == 1) then
         if (CRW(%nio-1)(1) == 1) then IO(%nio-1) = storage[IRW(%nio-1)]
                           else storage[IRW(%nio-1)] = IO(%nio-1);
```

D.28 MULT

```
MULT_generator(){
   GC_COMPILER_NAME:    MULT                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       2                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_MULT                 (GC_Operation)

   GC_NUM_STYLES:       3                       (GC_Int)
   GC_STYLE_LIST:       GC_ARRAY, GC_WALLACE_DADDA, GC_ITERATIVE
                                                (GC_MiscList)

   GC_MAX_PARAMS:       9                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
          (GC_LEFT_INPUT_WIDTH,  GC_Required, GC_Int,         %lw,
           GC_RIGHT_INPUT_WIDTH, GC_Required, GC_Int,         %rw,
           GC_STYLE,             GC_Required, GC_Misc,        %s,
           GC_PIPELINE_FLAG,     GC_Optional, GC_Boolean,     %pf,
           GC_PIPELINE_STAGES,   GC_Optional, GC_Int,         %ps,
           GC_DATA_TYPE,         GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,    GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,    GC_Optional, GC_EAttributeList,
                                 GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,    GC_Optional, GC_CharPtr      %af);
}

MULT_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%lw-1)],
                        I1[0-(%rw-1)]           (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%lw+rw-1)]        (GC_CharPtrPtrPtr)
}

MULT_component_op_to_pins((GENUS_operation) %o){
   GC_MULT:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
```

```
     GC_OP_INPUT_LIST:        GC_INPUT_LIST          (GC_CharPtrPtrPtr)
     GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS         (GC_Int)
     GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST         (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 * I1;
```

D.29 MUX

```
MUX_generator(){
   GC_COMPILER_NAME:    MUX                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_MAX_PARAMS:       7                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_INPUT,       GC_Required, GC_Int,         %n,
            GC_INVERT_FLAG,     GC_Optional, GC_Boolean,     %invert,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

MUX_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    %n                      (GC_Int)
   GC_FUNCTION_LIST:    I0-(%n-1)               (GC_CharPtrPtr)
   GC_NUM_CONTROLS:     %n                      (GC_Int)
   GC_CONTROL_LIST:     CI0-(%n-1)              (GC_CharPtrPtr)
}

MUX_component_op_to_pins((GC_CharPtr) %o){
  I%o:
     GC_OP_NUM_INPUTS:        1                      (GC_Int)
     GC_OP_INPUT_LIST:        GC_INPUT_LIST[%o]      (GC_CharPtrPtrPtr)
     GC_OP_NUM_OUTPUTS:       1                      (GC_Int)
     GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST[%o]     (GC_CharPtrPtrPtr)
     GC_OP_NUM_CONTROLS:      1                      (GC_Int)
     GC_OP_CONTROL_LIST:      GC_CONTROL_LIST[%o]    (GC_CharPtrPtr)
}

OPERATIONS :
        Select the input whose control input is 1.
        Assumption : Only one of the control input is 1.
```

D.30 NAND

```
NAND_generator(){
   GC_COMPILER_NAME:    NAND                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
```

```
   GC_FUNCTION_LIST:    GC_NAND                 (GC_Operation)
   GC_MAX_PARAMS:       6                       (GC_int)

   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

NAND_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

NAND_component_op_to_pins((GENUS_operation) %o){
   GC_NAND:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = not(I0 and I1 ... and I(n-1));
```

D.31 NOR

```
NOR_generator(){
   GC_COMPILER_NAME:    NOR                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_NOR                  (GC_Operation)

   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

NOR_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

NOR_component_op_to_pins((GENUS_operation) %o){
```

```
   GC_NOR:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = not(I0 or I1 ... or I(n-1));
```

D.32 ONESHOT

```
ONE_SHOT_generator(){
   GC_COMPILER_NAME:    ONE_SHOT                (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       0                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     1                       (GC_Int)
   GC_CONTROL_LIST:     CTRIGGER                (GC_CharPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_ONE_SHOT             (GC_Operation)

   GC_MAX_PARAMS:       5                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_PULSE_WIDTH,     GC_Required, GC_Int,         %w,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

ONE_SHOT_component_op_to_pins((GENUS_operation) %o){
   GC_ONE_SHOT:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
      GC_OP_NUM_OCNTROLS:       GC_NUM_CONTROLS         (GC_Int)
      GC_OP_CONTROL_LIST:       GC_COTROL_LIST          (GC_CharPtrPtr)
}

OPERATIONS :
        Generate a single pulse of width %w ns, whenever there is positive
        edge on the control line CTRIGGER.
```

D.33 OR

```
OR_generator(){
   GC_COMPILER_NAME:    OR                      (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_OR                   (GC_Operation)
```

```
   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

OR_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

OR_component_op_to_pins((GENUS_operation) %o){
   GC_OR:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 OR I1 ... OR I(n-1);
```

D.34 PARITY

```
PARITY_generator(){
   GC_COMPILER_NAME:    PARITY                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_OUTPUT_WIDTH:     1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0]                   (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_PARITY               (GC_Operation)
   GC_NUM_TYPES:        2                       (GC_Int)
   GC_TYPE_LIST:        (GC_ODD, GC_EVEN)       (GC_MiscList)
   GC_MAX_PARAMS:       6                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variable */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_TYPE,            GC_Required, GC_Misc,        %t,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

PARITY_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

PARITY_component_op_to_pins((GC_Operation)%o){
```

```
   GC_PARITY:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS   (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST   (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS  (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST  (GC_CharPtrPtrPtr)
}

OPERATIONS :
    ODD_PARITY:
        O0 = I0[0] xor I0[1] ... xor I0[%w-1];
    EVEN_PARITY:
        O0 = not(I0[0] xor I0[1] ... xor I0[%w-1]);
```

D.35 PORT

```
PORT_generator(){
   GC_COMPILER_NAME:    PORT                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)

   GC_NUM_MODES:        3                       (GC_Int)
   GC_MODE_LIST:        GC_INPUT, GC_OUTPUT, GC_BIDIRECTIONAL
                                                (GC_MiscList)

   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
          (GC_INPUT_WIDTH,      GC_Required, GC_Int,         %w,
           GC_MODE,             GC_Required, GC_Misc,        %m,
           GC_DATA_TYPE,        GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,   GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,   GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,   GC_Optional, GC_CharPtr      %af);
}

PORT_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)], O0[0-(%w-1)]  (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                          (GC_Int)
   GC_OUTPUT_LIST:      I0[0-(%w-1)], O0[0-(%w-1)]  (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     1                           (GC_Int)
   GC_CONTROL_LIST:     Cinput[0]                   (GC_CharPtrPtr)
}

OPERATIONS :
        INPUT : I0 = O0;
        OUTPUT : O0 = I0;
        INPUT_OUTPUT : If (Cinput = 1) I0 = O0; else O0 = I0;

I0 always gets connected to the internal components and O0 talks to
        the outside world. Cinput is the control line that decides the
        direction of data flow in INPUT_OUTPUT port.
```

D.36 REGFILE

```
REG_FILE_generator(){
   GC_COMPILER_NAME:    REG_FILE                (GC_CharPtr)
   GC_CLASS:            GC_Clocked              (GC_Class)
   GC_CLOCK:            CLK                     (GC_CharPtr)
```

```
   GC_NUM_ENABLES:      1                       (GC_Int)
   GC_ENABLE_LIST:      CEN                     (GC_CharPtr)

   GC_MAX_PARAMS:       10                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,        %iw,
            GC_NUM_WORDS,       GC_Required, GC_Int,        %nw,
            GC_NUM_INPUT_PORTS, GC_Required, GC_Int,        %ni,
            GC_NUM_OUTPUT_PORTS,GC_Required, GC_Int,        %no,
            GC_NUM_INOUT_PORTS, GC_Required, GC_Int,        %nio,
            GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,    %e,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

REG_FILE_component_initialize(){
   GC_NUM_INPUTS:       2 * %ni + 2 * %nio + %no    (GC_Int)
   GC_INPUT_LIST:       I0-(%ni-1)[0-(%iw-1)],
                        IO0-(%nio-1)[0-(%iw-1)],
                        IW0-(%ni-1)[0-(ln(%iw)-1)],
                        IR0-(%no-1)[0-(ln(%iw)-1)],
                        IRW0-(%nio-1)[0-(ln(%iw)-1)] (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                          (GC_Int)
   GC_NUM_OUTPUTS:      %no + %nio                  (GC_Int)
   GC_OUTPUT_LIST:      O0-(%no-1)[0-(%iw-1)],
                        IO0-(%nio-1)[0-(%iw-1)]     (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    %ni + %no + %nio
   GC_FUNCTION_LIST:    W0-(%ni-1), R0-(%no-1),
                        RW0-(%nio-1)                (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %ni + %no + %nio            (GC_Int)
   GC_CONTROL_LIST:     CW0-(%ni-1), CR0-(%no-1),
                        CRW0-1[0-(%nio-1)]          (GC_CharPtrPtrPtr)
}

REG_FILE_component_op_to_pins((GENUS_operation) %o){
   read_op(%o):
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[2*%ni+%nio+%o]
                                                        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%o]      (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%ni+%o] (GC_CharPtrPtrPtr)

   write_op(%o):
      GC_OP_NUM_INPUTS:         2                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[%o, %ni+%nio+%o]
                                                        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        0                       (GC_Int)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtrPtrPtr)

   read_write_op(%o):
      GC_OP_NUM_INPUTS:         2                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[%ni+%o, 2*%ni+%nio+%no+%o]
                                                        (GC_CharPtrPtrPtr)
```

```
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%no+%o]  (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%ni+%no+%o]
                                                        (GC_CharPtrPtrPtr)
}
OPERATIONS :
   WRITE :
      if (CW0 == 1) then storage[IW0] = I0;
      if (CW1 == 1) then storage[IW1] = I1;
                     :
                     :
                     :
      if (CW(%ni-1) == 1) then storage[IW(%ni-1)] = I(%ni-1);
   READ :
      if (CR0 == 1) then O0 = storage[IR0];
      if (CR1 == 1) then O1 = storage[IR1];
                     :
                     :
                     :
      if (CR(%no-1) == 1) then storage[IR(%no-1)] = I(%no-1);
   READ_WRITE :
      if (CRW0(0) == 1) then
         if (CRW0(1) == 1) then IO0 = storage[IRW0]
                            else storage[IRW0] = IO0;
      if (CRW1(0) == 1) then
         if (CRW1(1) == 1) then IO1 = storage[IRW1]
                            else storage[IRW1] = IO1;
                     :
                     :
                     :
      if (CRW(%nio-1)(0) == 1) then
         if (CRW(%nio-1)(1) == 1) then IO(%nio-1) = storage[IRW(%nio-1)]
                            else storage[IRW(%nio-1)] = IO(%nio-1);
```

D.37 REGISTER

```
REGISTER_generator(){
   GC_COMPILER_NAME:    REGISTER                (GC_CharPtr)
   GC_CLASS:            GC_Clocked              (GC_Class)
   GC_NUM_INPUTS:       4                       (GC_Int)
   GC_NUM_OUTPUTS:      4                       (GC_Int)
   GC_CLOCK:            CLK                     (GC_CharPtr)

   GC_NUM_ENABLES:      1                       (GC_Int)
   GC_ENABLE_LIST:      CEN                     (GC_CharPtr)

   GC_NUM_ASYNCS:       2                       (GC_Int)
   GC_ASYNC_LIST:       ASET, ARESET            (GC_CharPtrPtr)

   GC_NUM_CONTROLS:     4                       (GC_Int)
   GC_CONTROL_LIST:     CLOAD,CSHL,CSHR,CSCAN   (GC_CharPtrPtr)

   GC_NUM_FUNCTIONS:    4                       (GC_Int)
   GC_FUNCTION_LIST:    GC_LOAD,GC_SHL,GC_SHR,
                        GC_SCAN                 (GC_OperationList)

   GC_NUM_TYPES:        2                       (GC_Int)
```

```
   GC_TYPE_LIST:        GC_LATCH, GC_D_FF       (GC_MiscList)

   GC_MAX_PARAMS:       11                      (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,         %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,            %fl,
            GC_TYPE,            GC_Required, GC_Misc,        %type,
            GC_INVERT_OUT,      GC_Required, GC_Boolean,     %io,
            GC_SET_VALUE,       GC_Required, GC_Int,         %setv,
            GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,     %enable,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

REGISTER_component_initialize(){
   GC_INPUT_LIST:       IO[0-(%w-1)], ILIN[0], IRIN[0],
                        ISCAN[0]                        (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w
   GC_OUTPUT_LIST:      OO[0-(%w-1)],IROUT[0],ILOUT[0],
                        OSCAN[0]                        (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _REGISTER_control_list(%f, %fl) (GC_CharPtrPtr)
}

_REGISTER_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_LOAD => "CLOAD", GC_SHL => "CSHL", GC_SHR => "CSHR", GC_SCAN => "CSCAN"
}

REGISTER_component_op_to_pins((GC_Operation)%o){

   GC_LOAD:
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[0]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_SHL:
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[2]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        2                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,2]     (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)

   GC_SHR:
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[1]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        2                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[0,1]     (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                       (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]     (GC_CharPtr)
```

```
   GC_SCAN:
      GC_OP_NUM_INPUTS:          1                        (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST[3]         (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         2                        (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST[0,3]      (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:        1                        (GC_Int)
      GC_OP_CONTROL_LIST:        GC_CONTROL_LIST[%o]      (GC_CharPtr)
}

OPERATIONS :
     Synchronous behaviour
        if (io)
                LOAD : OON = not (IO);
                SHL : OLOUT, OON = not (shl (OO, IRIN));
                SHR : OON, OROUT = not (shr (OO, ILIN));
                SCAN : OON, OSCAN = not (shr (OO, ISCAN));
        else
                LOAD : OO = IO;
                SHL : OLOUT, OO = shl (OO, IRIN);
                SHR : OO, OROUT = shr (OO, ILIN);
                SCAN : OO, OSCAN = shr (OO, ISCAN);
     Asynchronous behaviour
        SET : OO = %setv;
        RESET : OO = 0;
```

D.38 ROM

```
ROM_generator(){
   GC_COMPILER_NAME:    ROM                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_ENABLES:      1                       (GC_Int)
   GC_ENABLE_LIST:      CEN                     (GC_CharPtr)

   GC_MAX_PARAMS:       9                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %iw,
            GC_NUM_WORDS,       GC_Required, GC_Int,         %nw,
            GC_NUM_OUTPUT_PORTS,GC_Required, GC_Int,         %no,
            GC_CORE_FILE,      ,GC_Required, GC_CharPtr,     %cf,
            GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,     %e,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

ROM_component_initialize(){
   GC_NUM_INPUTS:       %no                         (GC_Int)
   GC_INPUT_LIST:       IR0-(%no-1)[0-(ln(%iw)-1)], (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                          (GC_Int)
   GC_NUM_OUTPUTS:      %no                         (GC_Int)
   GC_OUTPUT_LIST:      O0-(%no-1)[0-(%iw-1)]       (GC_CharPtrPtrPtr)
   GC_NUM_FUNCTIONS:    %no                         (GC_Int)
   GC_FUNCTION_LIST:    R0-(%no-1)                  (GC_CharPtrPtrPtr)
}

ROM_component_op_to_pins((GC_CharPtr) %o){
```

```
  R%o:
      GC_OP_NUM_INPUTS:         1                       (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST[%o]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        1                       (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST[%o]      (GC_CharPtrPtrPtr)
}
OPERATIONS :
      O0 = %cf[IR0];
      O1 = %cf[IR1];
           :
           :
           :
      O(%no-1) = %cf[IR(%no-1)];
```

D.39 SELECTOR

```
SELECTOR_generator(){
   GC_COMPILER_NAME:    SELECTOR               (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial       (GC_Class)
   GC_NUM_OUTPUTS:      1                      (GC_Int)
   GC_NUM_FUNCTIONS:    1                      (GC_Int)
   GC_FUNCTION_LIST:    GC_SELECT              (GC_Operation)

   GC_MAX_PARAMS:       9                      (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
          (GC_INPUT_WIDTH,      GC_Required, GC_Int,         %w,
           GC_NUM_GUARDS,       GC_Required, GC_Int,         %n,
           GC_GUARD_LIST,       GC_Required, GC_EIntPtr,
                                GC_IntPtr,                   %gl,
           GC_ELSE_FLAG,        GC_Required, GC_Boolean,     %e,
           GC_CONTROL_WIDTH,    GC_Required, GC_Int,         %c,
           GC_DATA_TYPE,        GC_Optional, GC_Misc,        %d,
           GC_NUM_ATTRIBUTES,   GC_Optional, GC_Int,         %at,
           GC_ATTRIBUTE_LIST,   GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
           GC_ATTRIBUTE_FILE,   GC_Optional, GC_CharPtr      %af);
}

SELECTOR_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)], ISEL[0-(%c-1)],
                                               (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                     (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]           (GC_CharPtrPtrPtr)
}

SELECTOR_component_op_to_pins((GENUS_operation) %o){
   GC_SELECT:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        Selector is a generalized Multiplexer. In multiplexer, an input is
        selected if its corresponding control line is high. In selector, the
        inputs need not be in order. The value on ISEL is compared with the
        elements in the guard list serially. If it matches with the ith element
```

```
in the guard_list, then the ith input is selected. Also, we need not
have all the inputs present for given control_width. There is the
option of specifying else flag. If the flag is true, and if the
value on the ISEL does not match with any of the elements in the
guard_list, the last input is selected. Whenever flag is true,
there is an extra input. Appendix D contains an example of selector.
The guard_list contains two elements: 4 and 8 and else flag is true.
This selector performs following function:

        case ISEL is
             when 4 => O0 = I0;
             when 8 => O0 = I1;
             when others => O0 = I2;
end case;
```

D.40 SHIFTER

```
SHIFTER_generator(){

   GC_COMPILER_NAME:    SHIFTER                 (CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_INPUTS:       3                       (GC_Int)
   GC_NUM_OUTPUTS:      3                       (GC_Int)

   GC_NUM_CONTROLS:     4                       (GC_Int)
   GC_CONTROL_LIST:     CSHL, CSHR, CROTR, CROTL(GC_CharPtrPtr)

   GC_NUM_FUNCTIONS:    4                       (GC_Int)
   GC_FUNCTION_LIST:    GC_SHL, GC_SHR, GC_ROTL, GC_ROTL
                                                (GC_OperationList)

   GC_MAX_PARAMS:       7                       (GC_Int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,        %w,
            GC_NUM_FUNCTIONS,   GC_Required, GC_Int,        %f,
            GC_FUNCTION_LIST,   GC_Required, GC_EOperationList,
                                GC_OperationList,           %fl,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

SHIFTER_component_initialize(){

   GC_INPUT_LIST:       I0[0-(%w-1)], ILIN[0], IRIN[0]  (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                              (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)],OROUT[0],OLOUT[0]  (GC_CharPtrPtrPtr)
   GC_NUM_CONTROLS:     %f
   GC_CONTROL_LIST:     _SHIFTER_control_list(%f, %fl)  (GC_CharPtrPtr)
}

_SHIFTER_control_list((GC_Int)%f, (GC_OperationList)%fl){
   GC_SHL => "CSHL", GC_SHR => "CSHR", GC_ROTR => "CROTR",
   GC_ROTL => "CROTL",

}
```

```
SHIFTER_component_op_to_pins((GC_Operation)%o){

   GC_SHL:
      GC_OP_NUM_INPUTS:          2                        (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST[0,2]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         2                        (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST[0,2]      (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:        1                        (GC_Int)
      GC_OP_CONTROL_LIST:        GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_SHR:
      GC_OP_NUM_INPUTS:          2                        (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST[0,1]       (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         2                        (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST[0,1]      (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:        1                        (GC_Int)
      GC_OP_CONTROL_LIST:        GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_ROTL:
      GC_OP_NUM_INPUTS:          1                        (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST[0]         (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         1                        (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST[0]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:        1                        (GC_Int)
      GC_OP_CONTROL_LIST:        GC_CONTROL_LIST[%o]      (GC_CharPtr)

   GC_ROTL:
      GC_OP_NUM_INPUTS:          1                        (GC_Int)
      GC_OP_INPUT_LIST:          GC_INPUT_LIST[0]         (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:         1                        (GC_Int)
      GC_OP_OUTPUT_LIST:         GC_OUTPUT_LIST[0]        (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:        1                        (GC_Int)
      GC_OP_CONTROL_LIST:        GC_CONTROL_LIST[%o]      (GC_CharPtr
}

OPERATIONS :
        SHL : OLOUT, O0 = shl(I0, IRIN);
        SHR : O0, OROUT = shr(ILIN, I0);
        ROTL : O0 = rotl(I0);
        ROTR : O0 = rotr(I0);
```

D.41 STACK

```
STACK_generator(){
   GC_COMPILER_NAME:    STACK                   (GC_CharPtr)
   GC_CLASS:            GC_Clocked              (GC_Class)
   GC_NUM_INPUTS:       1                       (GC_Int)
   GC_NUM_OUTPUTS:      1                       (GC_Int)

   GC_NUM_CONTROLS:     2                       (GC_Int)
   GC_CONTROL_LIST      CPUSH, CPOP             (GC_CharPtrPtr)

   GC_CLOCK             CLK                     (GC_CharPtr)

   GC_NUM_ENABLES:      1                       (GC_Int)
   GC_ENABLE_LIST:      CEN                     (GC_CharPtr)
```

```
   GC_NUM_FUNCTIONS:    2                        (GC_Int)
   GC_FUNCTION_LIST:    GC_PUSH, GC_POP          (GC_OperationList)

   GC_MAX_PARAMS:       7                        (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_WORDS,       GC_Required, GC_Int,         %nw,
            GC_ENABLE_FLAG,     GC_Optional, GC_Boolean,     %e,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

STACK_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]             (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                       (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]             (GC_CharPtrPtrPtr)
   GC_CONTROL_LIST:     CPUSH, CPOP              (GC_CharPtrPtr)
}

STACK_component_op_to_pins((GENUS_operation) %o){
   GC_PUSH, GC_POP:
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS            (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST            (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS           (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROLS:       1                        (GC_Int)
      GC_OP_CONTROL_LIST:       GC_CONTROL_LIST[%o]      (GC_CharPtr)
}

OPERATIONS :
     Synchronous behaviour
        PUSH : Storage[top] = I0; top = top + 1;
        POP : O0 = Storage[top - 1]; top = top - 1;
     Assumption : No overflow or underflow;
```

D.42 TRISTATE

```
TRISATE_generator(){
   GC_COMPILER_NAME:    TRISATE                  (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial         (GC_Class)
   GC_NUM_INPUTS:       1                        (GC_Int)
   GC_NUM_OUTPUTS:      1                        (GC_Int)
   GC_NUM_FUNCTIONS:    1                        (GC_Int)
   GC_FUNCTION_LIST:    GC_TRISATE               (GC_Operation)
   GC_NUM_CONTROLS:     1                        (GC_Int)
   GC_CONTROL_LIST:     CPASS                    (GC_CharPtrPtr)

   GC_MAX_PARAMS:       6                        (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_INVERT_FLAG,     GC_Required, GC_Int,         %i,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
```

```
          GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

TRISATE_component_initialize(){
   GC_INPUT_LIST:       I0[0-(%w-1)]           (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                     (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]           (GC_CharPtrPtrPtr)
}

TRISATE_component_op_to_pins((GENUS_operation) %o){
   GC_TRISATE:
      GC_OP_COMMUTATIVE:       True                  (GC_Boolean)
      GC_OP_NUM_INPUTS:        GC_NUM_INPUTS         (GC_Int)
      GC_OP_INPUT_LIST:        GC_INPUT_LIST         (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:       GC_NUM_OUTPUTS        (GC_Int)
      GC_OP_OUTPUT_LIST:       GC_OUTPUT_LIST        (GC_CharPtrPtrPtr)
      GC_OP_NUM_CONTROL:       GC_NUM_CONTROLS       (GC_Int)
      GC_OP_CONTROL_LIST:      GC_CONTROL_LIST       (GC_CharPtrPtr)
}

OPERATIONS :
        if (CPASS = 1) then O0 = I0; else O0 = 'Z'; /* Not implemented yet */
```

D.43 USERDEFINED

```
USER_DEFINED_generator(){
   GC_COMPILER_NAME:    USER_DEFINED           (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial       (GC_Class)
   GC_EXTERNAL_PARAMETERS:      /* variables */
         (GC_COMPONENT_KEY,   GC_Required, GC_ComponentKey,%k,
          GC_NUM_INPUTS,      GC_Optional, GC_Int,        %ni,
          GC_INPUT_LIST,      GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %il,
          GC_NUM_OUTPUTS,     GC_Optional, GC_Int,        %no,
          GC_OUTPUT_LIST,     GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %ol,
          GC_NUM_INOUTS,      GC_Optional, GC_Int,        %nio,
          GC_INOUT_LIST,      GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %iol,
          GC_NUM_MISC_INPUTS, GC_Optional, GC_Int,        %nmi,
          GC_MISC_INPUT_LIST, GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %mil,
          GC_NUM_MISC_OUTPUTS,GC_Optional, GC_Int,        %nmo,
          GC_MISC_OUTPUT_LIST,GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %mol,
          GC_NUM_MISC_INOUTS, GC_Optional, GC_Int,        %nmio,
          GC_MISC_INOUT_LIST, GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %miol,
          GC_NUM_CLOCKS,      GC_Optional, GC_Int,        %ncl,
          GC_CLOCK_LIST,      GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %cll,
          GC_NUM_CONTROLS,    GC_Optional, GC_Int,        %ncn,
          GC_CONTROL_LIST,    GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %cnl,
          GC_NUM_ASYNCS,      GC_Optional, GC_Int,        %na,
          GC_ASYNC_LIST,      GC_Optional, GC_EPortList,
                              GC_CharPtrPtrPtr,           %al,
          GC_VHDL_FILE,       GC_Optional, GC_CharPtr,    %vf,
```

```
            GC_LOGIC_FILE,      GC_Optional, GC_CharPtr,     %lf,
            GC_DOC_FILE,        GC_Optional, GC_CharPtr,     %df,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

OPERATIONS :
        User defined;
```

D.44 WIREDOR

```
WIRED_OR_generator(){
   GC_COMPILER_NAME:    WIRED_OR                (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_WIRED_OR             (GC_Operation)

   GC_MAX_PARAMS:       7                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,         %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,         %n,
            GC_FAN_OUT,         GC_Required, GC_Int,         %f,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,        %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,         %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,            %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr      %af);
}

WIRED_OR_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

WIRED_OR_component_op_to_pins((GENUS_operation) %o){
   GC_WIRED_OR:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 or I1 ... or I(n-1);
```

D.45 XNOR

```
XNOR_generator(){
   GC_COMPILER_NAME:    XNOR                    (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
```

```
   GC_FUNCTION_LIST:    GC_XNOR                 (GC_Operation)

   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,        %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,        %n,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

XNOR_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

XNOR_component_op_to_pins((GENUS_operation) %o){
   GC_XNOR:
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = not(I0 xor I1 ... xor I(n-1));
```

D.46 XOR

```
XOR_generator(){
   GC_COMPILER_NAME:    XOR                     (GC_CharPtr)
   GC_CLASS:            GC_Combinatorial        (GC_Class)
   GC_NUM_OUTPUTS:      1                       (GC_Int)
   GC_NUM_FUNCTIONS:    1                       (GC_Int)
   GC_FUNCTION_LIST:    GC_XOR                  (GC_Operation)
   GC_MAX_PARAMS:       6                       (GC_int)
   GC_EXTERNAL_PARAMETERS:      /* variables */
           (GC_INPUT_WIDTH,     GC_Required, GC_Int,        %w,
            GC_NUM_INPUTS,      GC_Required, GC_Int,        %n,
            GC_DATA_TYPE,       GC_Optional, GC_Misc,       %d,
            GC_NUM_ATTRIBUTES,  GC_Optional, GC_Int,        %at,
            GC_ATTRIBUTE_LIST,  GC_Optional, GC_EAttributeList,
                                GC_AttributeList,           %atl,
            GC_ATTRIBUTE_FILE,  GC_Optional, GC_CharPtr     %af);
}

XOR_component_initialize(){
   GC_INPUT_LIST:       I0-(%n-1)[0-(%w-1)]     (GC_CharPtrPtrPtr)
   GC_OUTPUT_WIDTH:     %w                      (GC_Int)
   GC_OUTPUT_LIST:      O0[0-(%w-1)]            (GC_CharPtrPtrPtr)
}

XOR_component_op_to_pins((GENUS_operation) %o){
   GC_XOR:
```

```
      GC_OP_COMMUTATIVE:        True                    (GC_Boolean)
      GC_OP_NUM_INPUTS:         GC_NUM_INPUTS           (GC_Int)
      GC_OP_INPUT_LIST:         GC_INPUT_LIST           (GC_CharPtrPtrPtr)
      GC_OP_NUM_OUTPUTS:        GC_NUM_OUTPUTS          (GC_Int)
      GC_OP_OUTPUT_LIST:        GC_OUTPUT_LIST          (GC_CharPtrPtrPtr)
}

OPERATIONS :
        O0 = I0 xor I1 ... xor I(n-1);
```

E Functional Models for Components

E.1 ADDSUB

```
--------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                      GC_NUM_FUNCTIONS =  2
-- GC_FUNCTION_LIST = (GC_ADD, GC_SUB)     GC_STYLE = GC_RIPPLE_CARRY
--------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity ADD_SUB_11L1A2U3 is
port (I0, I1: in BIT_VECTOR(3 downto 0);
      ICIN : in BIT;
      CADD, CRSUB: in BIT;
      O0 : out BIT_VECTOR(3 downto 0);
      OCOUT, OVF : out BIT);
end ADD_SUB_11L1A2U3;

architecture ADD_SUB_11L1A2U3_behavior of ADD_SUB_11L1A2U3 is
begin
     process(I0, I1, ICIN, CADD, CRSUB)
          variable temp : BIT_VECTOR(4 downto 0);
          variable temp1 : BIT_VECTOR(3 downto 0);
          variable temp2 : BIT_VECTOR(3 downto 0);
          begin
               temp1 := I0;
               temp2 := I1;
               OVF <= '0';
               if (CADD = '1') then
                    temp := CARRY_ADD(temp1, temp2);
                    if (ICIN = '1') then
                         temp := INC(temp);
                    end if;
                    else
               if (CRSUB = '1') then
                    temp1 := ONES_COMP(temp1);
                    temp := CARRY_ADD(temp1, temp2);
                    if (ICIN = '1') then
                         temp := INC(temp);
                    end if;
               end if;
               end if;
               O0 <= temp(3 downto 0);
               OCOUT <= temp(4);
               if (((temp1(3) = '1') and (temp2(3) = '1') and (temp(4) = '0'))
                    or ((temp1(3) = '0') and (temp2(3) = '0') and (temp(4) = '1'))) then
                         OVF <= '1';
               else
                         OVF <= '0';
               end if;
          end process;
end ADD_SUB_11L1A2U3_behavior;

configuration ADD_SUB_11L1A2U3_config of ADD_SUB_11L1A2U3 is
     for ADD_SUB_11L1A2U3_behavior
     end for;
end ADD_SUB_11L1A2U3_config;
```

E.2 ALU

```
----------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                  GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (GC_ADD, GC_LNOT) GC_STYLE = GC_CARRY_LOOKAHEAD
----------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity ALU_11L1M113 is
port (I0, I1 : in BIT_VECTOR(3 downto 0);
      ICIN : in BIT;
      CADD, CLNOT : in BIT;
      O0 : out BIT_VECTOR(3 downto 0);
      O1, OCOUT, OZERO, OVF : out BIT);
end ALU_11L1M113;

architecture ALU_11L1M113_behavior of ALU_11L1M113 is
begin
     process (I0, I1, ICIN, CADD, CLNOT)
          variable ctrl_vec : BIT_VECTOR(1 downto 0);
          variable temp : BIT_VECTOR(4 downto 0);
          variable temp1 : BIT_VECTOR(3 downto 0);
          variable temp2 : BIT_VECTOR(3 downto 0);
          variable zero : BIT_VECTOR(3 downto 0);
          variable ctrl_int :  integer;
          begin
               ctrl_vec := CADD & CLNOT;
               temp1 := I0;
               temp2 := I1;
               OVF <= '0';
               ctrl_int := CRL_INTVAL(ctrl_vec);
               case ctrl_int is
                      when 0 => temp := CARRY_ADD(temp1, temp2);
                                if (ICIN = '1') then
                                   temp := INC(temp);
                                end if;
                                OCOUT <= temp(4);
                                O1 <= '0';
                      when 1 => temp(3 downto 0) := ONES_COMP (temp1);
                                OCOUT <= '0';
                                O1 <= '0';
                      when others => null;
               end case;

               O0 <= temp(3 downto 0);
               zero := temp(3 downto 0);
               if (B2I(zero) = 0) then
                    OZERO <= '1';
               else
                    OZERO <= '0';
               end if;
-- Checking overflow
               if (((temp1(3) = '1') and (temp2(3) = '1') and (temp(3) = '0'))
                    or ((temp1(3) = '0') and (temp2(3) = '0') and (temp(3) = '1'))) then
                         OVF <= '1';
               else
                         OVF <= '0';
               end if;
```

```
        end process;
end ALU_11L1M113_behavior;

configuration ALU_11L1M113_config of ALU_11L1M113 is
     for ALU_11L1M113_behavior
     end for;
end ALU_11L1M113_config;
```

E.3 AND

```
---------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                GC_NUM_INPUTS = 4
---------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity AND_Xt is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      I2 : in resMVL4_VECTOR(3 downto 0);
      I3 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end AND_Xt;

architecture AND_Xt_behavior of AND_Xt is
begin
     O0(0) <= I0(0) and I1(0) and I2(0) and I3(0);
     O0(1) <= I0(1) and I1(1) and I2(1) and I3(1);
     O0(2) <= I0(2) and I1(2) and I2(2) and I3(2);
     O0(3) <= I0(3) and I1(3) and I2(3) and I3(3);
end AND_Xt_behavior;

configuration AND_Xt_config of AND_Xt is
     for AND_Xt_behavior
     end for;
end AND_Xt_config;
```

E.4 BARRELSHIFTER

```
------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                 GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (SHL, ROTL)     GC_SHIFT_DISTANCE = 3
------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity BARREL_SHIFTER_Xov1W1 is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      ISHNUM : in resMVL4_VECTOR(1 downto 0);
      IFILL, CSHL, CROTL : in resMVL4;
      O0 : out resMVL4_VECTOR(3 downto 0));
end BARREL_SHIFTER_Xov1W1;

architecture BARREL_SHIFTER_Xov1W1_behavior of BARREL_SHIFTER_Xov1W1 is
begin
     process (I0, ISHNUM, IFILL, CSHL, CROTL)
```

```
          variable ctrl_vec : resMVL4_VECTOR(1 downto 0);
          variable ctrl_int, shft_num : integer;
          begin
               ctrl_vec := CSHL & CROTL;
               ctrl_int := CRL_INTVAL(ctrl_vec);
               shft_num := B2I(ISHNUM);

               assert (shft_num < I0'high+1)
               report "Invalid input values detected."
               severity note;

               if (shft_num < I0'high+1) then
                  case ctrl_int is
                      when 0 =>
                         if (IFILL = '1') then
                              O0 <= SHL1(I0, shft_num);
                         else
                              O0 <= SHL0(I0, shft_num);
                         end if;
                      when 1 => O0 <= ROTL(I0,shft_num);
                      when others => null;
                  end case;
               end if;
          end process;
end BARREL_SHIFTER_Xov1W1_behavior;

configuration BARREL_SHIFTER_Xov1W1_config of BARREL_SHIFTER_Xov1W1 is
     for BARREL_SHIFTER_Xov1W1_behavior
     end for;
end BARREL_SHIFTER_Xov1W1_config;
```

E.5 BUFFER

```
-------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
-------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity BUFFER_Xa is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0));
end BUFFER_Xa;

architecture BUFFER_Xa_behavior of BUFFER_Xa is
begin
       O0 <= not I0;
end BUFFER_Xa_behavior;

configuration BUFFER_Xa_config of BUFFER_Xa is
     for BUFFER_Xa_behavior
     end for;
end BUFFER_Xa_config;
```

E.6 BUS

```
------------------------------------------------------------
```

```
-- GC_INPUT_WIDTH = 4              GC_NUM_INPUTS = 6
-- GC_FAN_OUT = 4
-----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity BUS_XvQ is
port (I0, I1, I2, I3, I4, I5 : in resMVL4_VECTOR(3 downto 0);
      CI0, CI1, CI2, CI3, CI4, CI5 : in resMVL4;
      O0 : out resMVL4_VECTOR(3 downto 0));
end BUS_XvQ;

architecture BUS_XvQ_behavior of BUS_XvQ is
begin
     process (I0, I1, I2, I3, I4, I5, CI0, CI1, CI2, CI3, CI4, CI5)
     variable ctrl_vec : resMVL4_VECTOR(5 downto 0);
     variable ctrl_int : integer;
          begin
               ctrl_vec := CI0 & CI1 & CI2 & CI3 & CI4 & CI5;
               ctrl_int := CRL_INTVAL(ctrl_vec);
               case ctrl_int is
                    when 0 => O0 <= I0;
                    when 1 => O0 <= I1;
                    when 2 => O0 <= I2;
                    when 3 => O0 <= I3;
                    when 4 => O0 <= I4;
                    when 5 => O0 <= I5;
                    when others => O0 <= "ZZZZ";
               end case;
          end process;
end BUS_XvQ_behavior;

configuration BUS_XvQ_config of BUS_XvQ is
     for BUS_XvQ_behavior
     end for;
end BUS_XvQ_config;
```

E.7 CLOCKGEN

```
--------------------------------------------------------------
-- GC_CLOCK_PERIOD = 4             GC_CLOCK_HIGH = 2
--------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity CLOCK_GEN_A7 is
port (O0 : out resMVL4);
end CLOCK_GEN_A7;

architecture CLOCK_GEN_A7_behavior of CLOCK_GEN_A7 is
begin
   process
   begin
     loop
       O0 <= '1';
       wait for 2 ns;
       O0 <= '0';
       wait for 2 ns;
```

```
     end loop;
   end process;
end CLOCK_GEN_A7_behavior;

configuration CLOCK_GEN_A7_config of CLOCK_GEN_A7 is
     for CLOCK_GEN_A7_behavior
     end for;
end CLOCK_GEN_A7_config;
```

E.8 COMPARATOR

```
------------------------------------------------------------
-- GC_INPUT_WIDTH = 4              GC_NUM_FUNCTIONS = 3
-- GC_FUNCTION_LIST = (GC_NEQ, GC_LT, GC_LEQ)
------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity COMPARATOR_Xpo1 is
port (I0, I1 : in resMVL4_VECTOR(3 downto 0);
      ONEQ, OLT, OLEQ : out resMVL4);
end COMPARATOR_Xpo1;

architecture COMPARATOR_Xpo1_behavior of COMPARATOR_Xpo1 is
begin
     process (I0, I1)
     begin
          if (I0 /= I1) then ONEQ <= '1';
          else ONEQ <= '0';
          end if;
          if (I0 < I1) then OLT <= '1';
          else OLT <= '0';
          end if;
          if (I0 <= I1) then OLEQ <= '1';
          else OLEQ <= '0';
          end if;
     end process;

end COMPARATOR_Xpo1_behavior;

configuration COMPARATOR_Xpo1_config of COMPARATOR_Xpo1 is
     for COMPARATOR_Xpo1_behavior
     end for;
end COMPARATOR_Xpo1_config;
```

E.9 CONCAT

```
----------------------------------------------------------------
-- GC_NUM_INPUTS = 4          GC_INPUT_WIDTH_LIST = (4, 3, 5, 2)
----------------------------------------------------------------
entity CONCAT_ti is
port (
      I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(2 downto 0);
      I2 : in resMVL4_VECTOR(4 downto 0);
      I3 : in resMVL4_VECTOR(1 downto 0);
      O0 : out resMVL4_VECTOR(13 downto 0)     );
```

```
end CONCAT_ti;

architecture CONCAT_ti_behavior of CONCAT_ti is
begin

-- this architecture concats inputs in incremental order
-- that is I0 will be the left most and I'n' will be the
-- right most when output is created.

       O0 <= I0 & I1 & I2 & I3;
end CONCAT_ti_behavior;

configuration CONCAT_ti_config of CONCAT_ti is
     for CONCAT_ti_behavior
     end for;
end CONCAT_ti_config;
```

E.10 CONSTANT

```
------------------------------------------------------------------
-- GC_OUTPUT_WIDTH = 4                GC_CONST_VALUE = 7
-----------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity CONSTANT_E1H is
port (O0 : out resMVL4_VECTOR(3 downto 0)
      );
end CONSTANT_E1H;

architecture CONSTANT_E1H_behavior of CONSTANT_E1H is
begin
   O0 <= I2B(7,4);
end CONSTANT_E1H_behavior;

configuration CONSTANT_E1H_config of CONSTANT_E1H is
     for CONSTANT_E1H_behavior
     end for;
end CONSTANT_E1H_config;
```

E.11 COUNTER

```
---------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                   GC_NUM_FUNCTIONS = 3
-- GC_FUNCTION_LIST = (GC_LOAD, GC_COUNT_UP, GC_SCAN)
-- GC_TYPE = GC_BINARY                  GC_STYLE = GC_RIPPLE
-- GC_SET_VALUE = 9
---------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity COUNTER_11M193Z2T3Z2 is
port (I0 : in BIT_VECTOR(3 downto 0);
      ISCAN : in BIT;
      CLK, ASET, ARESET, CLOAD, CUP, CSCAN : in BIT;
      O0 : out BIT_VECTOR(3 downto 0);
      OSCAN : out BIT);
```

```
end COUNTER_11M193Z2T3Z2;

architecture COUNTER_11M193Z2T3Z2_behavior of COUNTER_11M193Z2T3Z2 is
begin
     process (ASET, ARESET, CLK)
          variable ctrl_vec : BIT_VECTOR(2 downto 0);
          variable storage  : BIT_VECTOR(3 downto 0);
          variable ctrl_int : integer;
          begin
               ctrl_vec := CLOAD & CUP & CSCAN;
                  if (ARESET = '1') then
                     storage := "0000";
                  else
                     if (ASET = '1') then
                        storage := I2B(9,4);
                     else
                        if (CLK = '1') then
                        ctrl_int := CRL_INTVAL(ctrl_vec);
                        case ctrl_int is
                           when 0 => storage := I0;
                           when 1 => if storage = "1111" then
                                        storage := "0000";
                                     else
                                        storage := INC(storage);
                                     end if;
                           when 2 => OSCAN <= storage(0);
                                    storage := SHR(storage, ISCAN);
                           when others => null;
                        end case;
                        end if;
                     end if;
                  end if;
                  O0 <= storage;
          end process;
end COUNTER_11M193Z2T3Z2_behavior;

configuration COUNTER_11M193Z2T3Z2_config of COUNTER_11M193Z2T3Z2 is
     for COUNTER_11M193Z2T3Z2_behavior
     end for;
end COUNTER_11M193Z2T3Z2_config;
```

E.12 DECODE

```
-----------------------------------------------------------
-- GC_INPUT_WIDTH = 4
----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity DECODE_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(15 downto 0));
end DECODE_X;

architecture DECODE_X_behavior of DECODE_X is
begin
     process (I0)
     begin
```

```
        for i in 0 to 15 loop
           if (i = B2I(I0)) then
              O0(i) <= '1';
           else
              O0(i) <= '0';
           end if;
        end loop;
     end process;
end DECODE_X_behavior;

configuration DECODE_X_config of DECODE_X is
     for DECODE_X_behavior
     end for;
end DECODE_X_config;
```

E.13 DELAY

```
----------------------------------------------------------
-- GC_DELAY_VALUE = 10          GC_INPUT_WIDTH = 4
----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity DELAY_OX is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0));
end DELAY_OX;

architecture DELAY_OX_behavior of DELAY_OX is
begin
       O0 <= I0 after 10 ns;
end DELAY_OX_behavior;

configuration DELAY_OX_config of DELAY_OX is
     for DELAY_OX_behavior
     end for;
end DELAY_OX_config;
```

E.14 DIV

```
------------------------------------------------------------
-- GC_INPUT_WIDTH = 4             GC_STYLE = GC_RESTORING
------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity DIV_X32 is
port (
      I0, I1: in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0));
end DIV_X32;

architecture DIV_X32_behavior of DIV_X32 is
begin
     process(I0,I1)
          begin
               O0 <= I0 / I1;
```

```
          end process;
end DIV_X32_behavior;

configuration DIV_X32_config of DIV_X32 is
     for DIV_X32_behavior
     end for;
end DIV_X32_config;
```

E.15 ENCODE

```
-----------------------------------------------------------
-- GC_INPUT_WIDTH = 4
-----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity ENCODE_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(1 downto 0));
end ENCODE_X;

architecture ENCODE_X_behavior of ENCODE_X is
begin
     process (I0)
          variable input_number: integer;
     begin
          input_number := 3 - crl_intval(I0);
          O0 <= I2B(input_number, 2);
     end process;
end ENCODE_X_behavior;

configuration ENCODE_X_config of ENCODE_X is
     for ENCODE_X_behavior
     end for;
end ENCODE_X_config;
```

E.16 EXTRACT

```
---------------------------------------------------------------
-- GC_INPUT_WIDTH = 10           GC_LEFT_INDEX = 8
-- GC_RIGHT_INDEX = 2
---------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity EXTRACT_djS1 is
port (I0 : in resMVL4_VECTOR(9 downto 0);
      O0 : out resMVL4_VECTOR(6 downto 0)
      );
end EXTRACT_djS1;

architecture EXTRACT_djS1_behavior of EXTRACT_djS1 is
begin
       O0 <= I0(8 downto 2);
end EXTRACT_djS1_behavior;

configuration EXTRACT_djS1_config of EXTRACT_djS1 is
```

```
     for EXTRACT_djS1_behavior
     end for;
end EXTRACT_djS1_config;
```

E.17 FIFO

```
------------------------------------------------------------
-- GC_INPUT_WIDTH = 4             GC_NUM_WORDS = 1024
-----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity FIFO_XVH is
port ( I0 : in resMVL4_VECTOR(3 downto 0);
       CLK, CPUSH, CPOP : in resMVL4;
       O0 : out resMVL4_VECTOR(3 downto 0));
end FIFO_XVH;


-- This design assumes that there is no over or under flow error.
-- i.e. Number of input storage will not exceed storage size,
--      or there will be no attemp to pop when storage is empty.

architecture FIFO_XVH_behavior of FIFO_XVH is
type resMVL4_array is array(integer range <>) of resMVL4_VECTOR(3 downto 0);
begin
     process(CLK)
          variable storage : resMVL4_array(1023 downto 0);
          variable head, tail : integer :=0;
          variable ctrl_int : integer ;
          variable ctrl_vec : resMVL4_VECTOR(1 downto 0);
          begin
               ctrl_vec := CPUSH & CPOP;
               if ((CLK) = '1') then
                    ctrl_int := CRL_INTVAL(ctrl_vec);
                    case ctrl_int is
                        when 0 => storage(head):= I0;
                                  head := (head + 1) mod 1024;
                        when 1 => O0 <= storage(tail);
                                  tail := (tail + 1) mod 1024;
                        when others => null;
                    end case;
               end if;
          end process;
end FIFO_XVH_behavior;

configuration FIFO_XVH_config of FIFO_XVH is
     for FIFO_XVH_behavior
     end for;
end FIFO_XVH_config;
```

E.18 GAND

```
-------------------------------------------------------
-- GC_INPUT_WIDTH = 4
------------------------------------------------------
use work.types.all;
```

```
use work.resMVL4_FUNCTIONS.all;

entity GAND_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4);
end GAND_X;

architecture GAND_X_behavior of GAND_X is
begin
      O0 <= I0(0) and I0(1) and I0(2) and I0(3);
end GAND_X_behavior;

configuration GAND_X_config of GAND_X is
     for GAND_X_behavior
     end for;
end GAND_X_config;
```

E.19 GNAND

```
---------------------------------------------------------
-- GC_INPUT_WIDTH = 4
---------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity GNAND_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4);
end GNAND_X;

architecture GNAND_X_behavior of GNAND_X is
begin
      O0 <= not ( I0(0) and I0(1) and I0(2) and I0(3) );
end GNAND_X_behavior;

configuration GNAND_X_config of GNAND_X is
     for GNAND_X_behavior
     end for;
end GNAND_X_config;
```

E.20 GNOR

```
----------------------------------------------------------
-- GC_INPUT_WIDTH = 4
----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity GNOR_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4);
end GNOR_X;

architecture GNOR_X_behavior of GNOR_X is
begin
      O0 <= not ( I0(0) or I0(1) or I0(2) or I0(3) );
end GNOR_X_behavior;
```

```
configuration GNOR_X_config of GNOR_X is
     for GNOR_X_behavior
     end for;
end GNOR_X_config;
```

E.21 GNOT

```
--------------------------------------------------------------
--------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity GNOT is
port (I0 : in resMVL4;
      O0 : out resMVL4);
end GNOT;

architecture GNOT_behavior of GNOT is
begin
      O0 <= not I0;
end GNOT_behavior;

configuration GNOT_config of GNOT is
     for GNOT_behavior
     end for;
end GNOT_config;
```

E.22 GOR

```
-------------------------------------------------------
-- GC_INPUT_WIDTH = 2
-------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity GOR_V is
port (I0 : in resMVL4_VECTOR(1 downto 0);
      O0 : out resMVL4);
end GOR_V;

architecture GOR_V_behavior of GOR_V is
begin
      O0 <= I0(0) or I0(1);
end GOR_V_behavior;

configuration GOR_V_config of GOR_V is
     for GOR_V_behavior
     end for;
end GOR_V_config;
```

E.23 GXNOR

```
-------------------------------------------------------
-- GC_INPUT_WIDTH = 4
-------------------------------------------------------
use work.types.all;
```

```
use work.resMVL4_FUNCTIONS.all;

entity GXNOR_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4);
end GXNOR_X;

architecture GXNOR_X_behavior of GXNOR_X is
begin
      O0 <= not ( I0(0) xor I0(1) xor I0(2) xor I0(3) );
end GXNOR_X_behavior;

configuration GXNOR_X_config of GXNOR_X is
     for GXNOR_X_behavior
     end for;
end GXNOR_X_config;
```

E.24 GXOR

```
----------------------------------------------------
-- GC_INPUT_WIDTH = 8
----------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity GXOR_b is
port (I0 : in resMVL4_VECTOR(7 downto 0);
      O0 : out resMVL4);
end GXOR_b;

architecture GXOR_b_behavior of GXOR_b is
begin
      O0 <= I0(0) xor I0(1) xor I0(2) xor I0(3) xor
            I0(4) xor I0(5) xor I0(6) xor I0(7);
end GXOR_b_behavior;

configuration GXOR_b_config of GXOR_b is
     for GXOR_b_behavior
     end for;
end GXOR_b_config;
```

E.25 LU

```
-------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                  GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (GC_ZERO, GC_RNOT)
-------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity LU_11L1M3 is
port (I0, I1 : in BIT_VECTOR(3 downto 0);
      CZERO, CRNOT : in BIT;
      O0 : out BIT_VECTOR(3 downto 0));
end LU_11L1M3;

architecture LU_11L1M3_behavior of LU_11L1M3 is
```

```
begin
     process (I0, I1, CZERO, CRNOT)
          variable ctrl_vec : BIT_VECTOR(1 downto 0);
          variable ctrl_int :  integer;
          begin
               ctrl_vec := CZERO & CRNOT;
               ctrl_int := CRL_INTVAL(ctrl_vec);
               case ctrl_int is
                      when 0 => O0 <=  "0000";
                      when 1 => O0 <= not (I1);
                      when others => null;
               end case;
          end process;
end LU_11L1M3_behavior;

configuration LU_11L1M3_config of LU_11L1M3 is
     for LU_11L1M3_behavior
     end for;
end LU_11L1M3_config;
```

E.26 MEMORY

```
----------------------------------------------------------------------------
-- GC_INPUT_WIDTH = 3                          GC_NUM_INPUT_PORTS = 1
-- GC_NUM_OUTPUT_PORTS = 2                     GC_NUM_INOUT_PORTS = 3
-- GC_NUM_WORDS = 30
---------------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity MEMORY_01P2O1T1O1B1 is
port (I0 : in resMVL4_VECTOR(2 downto 0);
      IO0, IO1, IO2 : inout resMVL4_VECTOR(2 downto 0);
      IW0 : in resMVL4_VECTOR(4 downto 0);
      IR0, IR1 : in resMVL4_VECTOR(4 downto 0);
      IRW0, IRW1, IRW2 : in resMVL4_VECTOR(4 downto 0);
      CLK : in resMVL4;
      CW0 : in resMVL4;
      CR0, CR1 : in resMVL4;
      CRW0, CRW1, CRW2 : in resMVL4_VECTOR(1 downto 0);
      O0, O1 : out resMVL4_VECTOR(2 downto 0));
end MEMORY_01P2O1T1O1B1;


-- This design assumes that there is no addressing error.
-- i.e. No two input address is addressing the same storage address,
--      or there is no input addressing nonexisting address.

architecture MEMORY_01P2O1T1O1B1_behavior of MEMORY_01P2O1T1O1B1 is
type data_array is array(integer range <>) of resMVL4_VECTOR(2 downto 0);
begin
     process(CLK, CR0, CR1)
          variable storage : data_array(29 downto 0);
          begin
               if ((CLK) = '1') then
                    if (CW0 = '1') then storage(B2I(IW0)):= I0; end if;
                    if (CRW0(1) = '1') then
                         if (CRW0(0) = '1') then
```

```
                              IO0 <= storage(B2I(IRW0));
                         else
                              storage(B2I(IRW0)):= IO0;
                         end if;
                    else
                         IO0 <= "ZZZ";
                    end if;
                    if (CRW1(1) = '1') then
                         if (CRW1(0) = '1') then
                              IO1 <= storage(B2I(IRW1));
                         else
                              storage(B2I(IRW1)):= IO1;
                         end if;
                    else
                         IO1 <= "ZZZ";
                    end if;
                    if (CRW2(1) = '1') then
                         if (CRW2(0) = '1') then
                              IO2 <= storage(B2I(IRW2));
                         else
                              storage(B2I(IRW2)):= IO2;
                         end if;
                    else
                         IO2 <= "ZZZ";
                    end if;
               end if;
               if (CR0 = '1') then O0 <= storage(B2I(IR0));end if;
               if (CR1 = '1') then O1 <= storage(B2I(IR1));end if;
          end process;
end MEMORY_01P2O1T1O1B1_behavior;

configuration MEMORY_01P2O1T1O1B1_config of MEMORY_01P2O1T1O1B1 is
     for MEMORY_01P2O1T1O1B1_behavior
     end for;
end MEMORY_01P2O1T1O1B1_config;
```

E.27 MULT

```
---------------------------------------------------------------------------
-- GC_LEFT_INPUT_WIDTH = 6               GC_RIGHT_INPUT_WIDTH = 4
-- GC_STYLE = GC_ARRAY
---------------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity MULT_C1S2U2B1 is
port (I0 : in resMVL4_VECTOR(5 downto 0);
      I1: in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(9 downto 0));
end MULT_C1S2U2B1;

architecture MULT_C1S2U2B1_behavior of MULT_C1S2U2B1 is
begin
     process(I0,I1)
          variable temp0, temp1, temp : resMVL4_VECTOR(9 downto 0);
          begin
               if (I0(5) = '1') then
                    temp0 := "0000" & TWOs_COMP(I0);
```

```
            else
                temp0 := "0000" & I0;
            end if;
            if (I1(3) = '1') then
                temp1 := "000000" & TWOs_COMP(I1);
            else
                temp1 := "000000" & I1;
            end if;
            temp := temp0 * temp1;
            if (((I0(5) = '1') and (I1(3) = '0')) or ((I0(5) = '0')
                and (I1(3) = '1'))) then
                O0 <= TWOs_COMP(temp);
            else
                O0 <= temp;
            end if;
        end process;
end MULT_C1S2U2B1_behavior;

configuration MULT_C1S2U2B1_config of MULT_C1S2U2B1 is
    for MULT_C1S2U2B1_behavior
    end for;
end MULT_C1S2U2B1_config;
```

E.28 MUX

```
------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                    GC_NUM_INPUTS = 3
------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity MUX_Xs is
port (I0, I1, I2 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0);
      CI0, CI1, CI2 : in resMVL4);
end MUX_Xs;

architecture MUX_Xs_behavior of MUX_Xs is
begin
    process (I0, I1, I2, CI0, CI1, CI2)
    variable ctrl_vec : resMVL4_VECTOR(2 downto 0);
    variable temp : resMVL4_VECTOR(3 downto 0);
    variable ctrl_int : integer;
        begin
            ctrl_vec := CI0 & CI1 & CI2;
            ctrl_int := CRL_INTVAL(ctrl_vec);
            case ctrl_int is
                when 0 => O0 <= I0;
                when 1 => O0 <= I1;
                when 2 => O0 <= I2;
                when others => null;
            end case;
        end process;
end MUX_Xs_behavior;

configuration MUX_Xs_config of MUX_Xs is
    for MUX_Xs_behavior
    end for;
```

```
end MUX_Xs_config;
```

E.29 NAND

```
-----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                    GC_NUM_INPUTS = 4
-----------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity NAND_Xt is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      I2 : in resMVL4_VECTOR(3 downto 0);
      I3 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end NAND_Xt;

architecture NAND_Xt_behavior of NAND_Xt is
begin
   O0(0) <= not( I0(0) and I1(0) and I2(0) and I3(0));
   O0(1) <= not( I0(1) and I1(1) and I2(1) and I3(1));
   O0(2) <= not( I0(2) and I1(2) and I2(2) and I3(2));
   O0(3) <= not( I0(3) and I1(3) and I2(3) and I3(3));
end NAND_Xt_behavior;

configuration NAND_Xt_config of NAND_Xt is
     for NAND_Xt_behavior
     end for;
end NAND_Xt_config;
```

E.30 NOR

```
----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                   GC_NUM_INPUTS = 4
----------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity NOR_Xt is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      I2 : in resMVL4_VECTOR(3 downto 0);
      I3 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end NOR_Xt;

architecture NOR_Xt_behavior of NOR_Xt is
begin
   O0(0) <= not( I0(0) or I1(0) or I2(0) or I3(0));
   O0(1) <= not( I0(1) or I1(1) or I2(1) or I3(1));
   O0(2) <= not( I0(2) or I1(2) or I2(2) or I3(2));
   O0(3) <= not( I0(3) or I1(3) or I2(3) or I3(3));
end NOR_Xt_behavior;
```

```
configuration NOR_Xt_config of NOR_Xt is
     for NOR_Xt_behavior
     end for;
end NOR_Xt_config;
```

E.31 ONESHOT

```
----------------------------------------------------------------
-- GC_PULSE_WIDTH = 4
----------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity ONE_SHOT_S1 is
port (CTRIGGER : in resMVL4;
      O0 : out resMVL4);
end ONE_SHOT_S1;

architecture ONE_SHOT_S1_behavior of ONE_SHOT_S1 is
begin
   process
   begin
       if ( CTRIGGER = '1') and (CTRIGGER'event) then
          O0 <= '1';
          wait for 4 ns;
          O0 <= '0';
       end if;
       wait on CTRIGGER;
   end process;
end ONE_SHOT_S1_behavior;

configuration ONE_SHOT_S1_config of ONE_SHOT_S1 is
     for ONE_SHOT_S1_behavior
     end for;
end ONE_SHOT_S1_config;
```

E.32 OR

```
-------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                 GC_NUM_INPUTS = 2
-------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity OR_Xr is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end OR_Xr;

architecture OR_Xr_behavior of OR_Xr is
begin
   O0(0) <= I0(0) or I1(0);
   O0(1) <= I0(1) or I1(1);
   O0(2) <= I0(2) or I1(2);
   O0(3) <= I0(3) or I1(3);
```

```
end OR_Xr_behavior;

configuration OR_Xr_config of OR_Xr is
     for OR_Xr_behavior
     end for;
end OR_Xr_config;
```

E.33 PARITY

```
--------------------------------------------------------
-- GC_INPUT_WIDTH = 4            GC_TYPE = GC_EVEN
--------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity PARITY_Xm1 is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4);
end PARITY_Xm1;

architecture PARITY_Xm1_behavior of PARITY_Xm1 is
begin
      O0 <= not(I0(0) xor I0(1) xor I0(2) xor I0(3));
end PARITY_Xm1_behavior;

configuration PARITY_Xm1_config of PARITY_Xm1 is
     for PARITY_Xm1_behavior
     end for;
end PARITY_Xm1_config;
```

E.34 PORT

```
--------------------------------------------------------
-- GC_INPUT_WIDTH = 4     GC_MODE = GC_INPUT_OUTPUT
--------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity PORT_X11 is
port (I0, O0 : inout resMVL4_VECTOR(3 downto 0);
      Cinput : in resMVL4);
end PORT_X11;

architecture PORT_X11_behavior of PORT_X11 is
begin
   process(Cinput, I0, O0)
   begin
     if (Cinput = '1') then
       I0 <= O0;
       O0 <= "ZZZZ";
     else
       O0 <= I0;
       I0 <= "ZZZZ";
     end if;
   end process;
end PORT_X11_behavior;
```

```
configuration PORT_X11_config of PORT_X11 is
     for PORT_X11_behavior
     end for;
end PORT_X11_config;
```

E.35 REGFILE

```
-----------------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                    GC_NUM_WORDS = 1024
-- GC_NUM_INPUT_PORTS = 1                GC_NUM_OUTPUT_PORTS = 2
-- GC_NUM_INOUT_PORTS = 2
-----------------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity REG_FILE_11BUO1T1N1B1 is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      IO0, IO1 : inout resMVL4_VECTOR(3 downto 0);
      IW0 : in resMVL4_VECTOR(9 downto 0);
      IR0, IR1 : in resMVL4_VECTOR(9 downto 0);
      IRW0, IRW1 : in resMVL4_VECTOR(9 downto 0);
      CLK : in resMVL4;
      CW0 : in resMVL4;
      CR0, CR1 : in resMVL4;
      CRW0, CRW1 : in resMVL4_VECTOR(1 downto 0);
      O0, O1 : out resMVL4_VECTOR(3 downto 0));
end REG_FILE_11BUO1T1N1B1;


-- This design assumes that there is no addressing error.
-- i.e. No two input address is addressing the same storage address,
--      or there is no input addressing nonexisting address.

architecture REG_FILE_11BUO1T1N1B1_behavior of REG_FILE_11BUO1T1N1B1 is
type data_array is array(integer range <>) of resMVL4_VECTOR(3 downto 0);
signal storage : data_array(1023 downto 0);
begin
     process(CLK)
     begin
          if (not CLK'Stable and CLK = '1') then
               if (CW0 = '1') then storage(B2I(IW0)) <= I0; end if;
               if (CRW0(0) = '1') then
                    if (CRW0(1) = '0') then storage(B2I(IRW0)) <= IO0; end if;
               end if;
               if (CRW1(0) = '1') then
                    if (CRW1(1) = '0') then storage(B2I(IRW1)) <= IO1; end if;
               end if;
          end if;
     end process;

     IO0 <= storage(B2I(IRW0))
          when CRW0(0) = '1' and CRW0(1) = '1' else
          "ZZZZ";

     IO1 <= storage(B2I(IRW1))
          when CRW1(0) = '1' and CRW1(1) = '1' else
          "ZZZZ";
```

```
     R0 : block (CR0 = '1')
     begin
          O0 <= guarded storage(B2I(IR0));
     end block R0;

     R1 : block (CR1 = '1')
     begin
          O1 <= guarded storage(B2I(IR1));
     end block R1;

end REG_FILE_11BUO1T1N1B1_behavior;

configuration REG_FILE_11BUO1T1N1B1_config of REG_FILE_11BUO1T1N1B1 is
     for REG_FILE_11BUO1T1N1B1_behavior
     end for;
end REG_FILE_11BUO1T1N1B1_config;
```

E.36 REGISTER

```
----------------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                       GC_NUM_FUNCTIONS = 3
-- GC_FUNCTION_LIST = (GC_LOAD, GC_SHR, GC_SCAN)
-- GC_TYPE = GC_D_FF                        GC_SET_VALUE = 10
-- GC_INVERT_OUT = True
----------------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity REGISTER_11M1B4835103 is
port (I0 : in BIT_VECTOR(3 downto 0);
      ILIN, ISCAN, CLK, ASET, ARESET, CLOAD, CSHR, CSCAN : in BIT;
      O0N : out BIT_VECTOR(3 downto 0);
      OROUT, OSCAN : out BIT);
end REGISTER_11M1B4835103;

architecture REGISTER_11M1B4835103_behavior of REGISTER_11M1B4835103 is
begin
     process (ASET, ARESET, CLK)
          variable ctrl_vec : BIT_VECTOR(2 downto 0);
          variable storage  : BIT_VECTOR(3 downto 0);
          variable ctrl_int : integer;
          begin
               ctrl_vec := CLOAD & CSHR & CSCAN;
                  if (ARESET = '1') then
                     storage := "0000";
                  else
                     if (ASET = '1') then
                        storage := I2B(10,4);
                     else
                      if (CLK = '1') then
                        ctrl_int := CRL_INTVAL(ctrl_vec);
                        case ctrl_int is
                           when 0 => storage := I0;
                           when 1 => OROUT <= storage(0);
                                     storage := SHR(storage, ILIN);
                           when 2 => OSCAN <= storage(0);
                                     storage := SHR(storage, ISCAN);
                           when others => null;
```

```
                    end case;
                  end if;
                 end if;
              end if;
              OON <= not(storage);
       end process;
end REGISTER_11M1B4835103_behavior;

configuration REGISTER_11M1B4835103_config of REGISTER_11M1B4835103 is
     for REGISTER_11M1B4835103_behavior
     end for;
end REGISTER_11M1B4835103_config;
```

E.37 ROM

```
----------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                     GC_NUM_WORDS = 7
-- GC_NUM_OUTPUT_PORTS = 3                GC_CORE_FILE = "core_file"
----------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity ROM_X61xD is
port (IR0, IR1, IR2 : in resMVL4_VECTOR(2 downto 0);
      O0, O1, O2 : out resMVL4_VECTOR(3 downto 0));
end ROM_X61xD;

architecture ROM_X61xD_behavior of ROM_X61xD is
type resMVL4_array is array(integer range <>) of resMVL4_VECTOR(3 downto 0);
CONSTANT storage : resMVL4_array(6 downto 0) :=
(('0', '0', 'X', '0'),
('1', '1', '0', '0'),
('0', '0', 'X', 'X'),
('1', '1', '0', '0'),
('0', '0', 'X', 'X'),
('0', '0', '0', '0'),
('1', '1', '1', '1'));
begin
     O0 <= storage(B2I(IR0));
     O1 <= storage(B2I(IR1));
     O2 <= storage(B2I(IR2));
end ROM_X61xD_behavior;

configuration ROM_X61xD_config of ROM_X61xD is
     for ROM_X61xD_behavior
     end for;
end ROM_X61xD_config;
```

E.38 SELECTOR

```
--------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                GC_NUM_GUARDS = 4
-- GC_GUARD_LIST = (4, 8)            GC_CONTROL_WIDTH = 4
-- GC_ELSE_FLAG = True
-------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;
```

```
entity SELECTOR_XpcJG is
port (I0, I1, I2 : in resMVL4_VECTOR(3 downto 0);
      ISEL : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0));
end SELECTOR_XpcJG;

architecture SELECTOR_XpcJG_behavior of SELECTOR_XpcJG is
begin
     process (ISEL, I0, I1, I2)
          variable ctrl_int : integer;
     begin
          ctrl_int := B2I(ISEL);
          case ctrl_int is
               when 4 => O0 <= I0;
               when 8 => O0 <= I1;
               when others => O0 <= I2 ;
          end case;
     end process;
end SELECTOR_XpcJG_behavior;

configuration SELECTOR_XpcJG_config of SELECTOR_XpcJG is
     for SELECTOR_XpcJG_behavior
     end for;
end SELECTOR_XpcJG_config;
```

E.39 SHIFTER

```
---------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                    GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (SHL, ROTL)
---------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity SHIFTER_Xov1 is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      IRIN, CSHL, CROTL : in resMVL4;
      O0 : out resMVL4_VECTOR(3 downto 0));
end SHIFTER_Xov1;

architecture SHIFTER_Xov1_behavior of SHIFTER_Xov1 is
begin
     process (I0, IRIN, CSHL, CROTL)
          variable ctrl_vec : resMVL4_VECTOR(1 downto 0);
          variable ctrl_int :  integer;
          begin
               ctrl_vec := CSHL & CROTL;
               ctrl_int := CRL_INTVAL(ctrl_vec);
               case ctrl_int is
                      when 0 => O0 <= I0(2 downto 0) & IRIN;
                      when 1 => O0 <= I0(2 downto 0) & I0(3);
                      when others => null;
               end case;
          end process;
end SHIFTER_Xov1_behavior;

configuration SHIFTER_Xov1_config of SHIFTER_Xov1 is
```

```
     for SHIFTER_Xov1_behavior
     end for;
end SHIFTER_Xov1_config;
```

E.40 STACK

```
----------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                          GC_NUM_WORDS = 2
----------------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity STACK_X11 is
port ( I0 : in resMVL4_VECTOR(3 downto 0);
       CLK, CPUSH, CPOP : in resMVL4;
       O0 : out resMVL4_VECTOR(3 downto 0));
end STACK_X11;


-- This design assumes that there is no over or under flow error.
-- i.e. Number of input storage will not exceed storage size,
--      or there will be no attemp to pop when storage is empty.

architecture STACK_X11_behavior of STACK_X11 is
type resMVL4_array is array(integer range <>) of resMVL4_VECTOR(3 downto 0);
begin
     process(CLK)
          variable storage : resMVL4_array(1 downto 0);
          variable top : integer :=0;
          variable ctrl_int : integer ;
          variable ctrl_vec : resMVL4_VECTOR(1 downto 0);
          begin
               ctrl_vec := CPUSH & CPOP;
               if ((CLK) = '1') then
                    ctrl_int := CRL_INTVAL(ctrl_vec);
                    case ctrl_int is
                        when 0 => storage(top):= I0;
                                  top := (top + 1) mod 2;
                        when 1 => top := (top + 1) mod 2;
                                  O0 <= storage(top);
                        when others => null;
                    end case;
               end if;
          end process;
end STACK_X11_behavior;

configuration STACK_X11_config of STACK_X11 is
     for STACK_X11_behavior
     end for;
end STACK_X11_config;
```

E.41 TRISTATE

```
--------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
--------------------------------------------------------------------
use work.types.all;
```

```
use work.resMVL4_FUNCTIONS.all;

entity TRISTATE_X is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      CPASS : in resMVL4;
      O0 : out resMVL4_VECTOR(3 downto 0)
      );
end TRISTATE_X;

architecture TRISTATE_X_behavior of TRISTATE_X is
begin
   process(CPASS, I0)
   begin
     if (CPASS = '1') then
       O0 <= I0;
     else
       O0 <= "ZZZZ";
     end if;
   end process;
end TRISTATE_X_behavior;

configuration TRISTATE_X_config of TRISTATE_X is
     for TRISTATE_X_behavior
     end for;
end TRISTATE_X_config;
```

E.42 WIREDOR

```
----------------------------------------------------------
-- GC_INPUT_WIDTH = 4                   GC_NUM_INPUTS = 2
-- GC_FAN_OUT = 4
----------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity WIRED_OR_XrQ is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end WIRED_OR_XrQ;

architecture WIRED_OR_XrQ_behavior of WIRED_OR_XrQ is
begin
   O0(0) <= I0(0) or I1(0);
   O0(1) <= I0(1) or I1(1);
   O0(2) <= I0(2) or I1(2);
   O0(3) <= I0(3) or I1(3);
end WIRED_OR_XrQ_behavior;

configuration WIRED_OR_XrQ_config of WIRED_OR_XrQ is
     for WIRED_OR_XrQ_behavior
     end for;
end WIRED_OR_XrQ_config;
```

E.43 XNOR

```
--------------------------------------------------------------
```

```
-- GC_INPUT_WIDTH = 4                GC_NUM_INPUTS = 2
---------------------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity XNOR_Xr is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end XNOR_Xr;

architecture XNOR_Xr_behavior of XNOR_Xr is
begin
   O0(0) <= not( I0(0) xor I1(0));
   O0(1) <= not( I0(1) xor I1(1));
   O0(2) <= not( I0(2) xor I1(2));
   O0(3) <= not( I0(3) xor I1(3));
end XNOR_Xr_behavior;

configuration XNOR_Xr_config of XNOR_Xr is
     for XNOR_Xr_behavior
     end for;
end XNOR_Xr_config;
```

E.44 XOR

```
-----------------------------------------------------
-- GC_INPUT_WIDTH = 4         GC_NUM_INPUTS = 2
----------------------------------------------------
use work.types.all;
use work.resMVL4_FUNCTIONS.all;

entity XOR_Xr is
port (I0 : in resMVL4_VECTOR(3 downto 0);
      I1 : in resMVL4_VECTOR(3 downto 0);
      O0 : out resMVL4_VECTOR(3 downto 0)
     );
end XOR_Xr;

architecture XOR_Xr_behavior of XOR_Xr is
begin
   O0(0) <= I0(0) xor I1(0);
   O0(1) <= I0(1) xor I1(1);
   O0(2) <= I0(2) xor I1(2);
   O0(3) <= I0(3) xor I1(3);
end XOR_Xr_behavior;

configuration XOR_Xr_config of XOR_Xr is
     for XOR_Xr_behavior
     end for;
end XOR_Xr_config;
```

F Synthesis Models for Components

F.1 ADDSUB

```
------------------------------------------------------------------
--    GC_INPUT_WIDTH = 4,                  GC_NUM_FUNCTIONS = 2,
--    GC_FUNCTION_LIST = (GC_ADD, GC_SUB)  GC_STYLE = GC_RIPPLE_CARRY
------------------------------------------------------------------
INORDER = CADD[0] CRSUB[0] I0[0] I0[1] I0[2] I0[3]
          I1[0] I1[1] I1[2] I1[3] ICIN[0];
OUTORDER = O0[0] O0[1] O0[2] O0[3] OCOUT[0] OVF[0];

T0[0] = I0[0] ^ CRSUB[0];
T1[0] = I1[0];
T0[1] = I0[1] ^ CRSUB[0];
T1[1] = I1[1];
T0[2] = I0[2] ^ CRSUB[0];
T1[2] = I1[2];
T0[3] = I0[3] ^ CRSUB[0];
T1[3] = I1[3];

C0 = ICIN[0];
C1 = (T0[0] * T1[0]) + (ICIN[0] * (T0[0] + T1[0]));
C2 = (T0[1] * T1[1]) + (C1 * (T0[1] + T1[1]));
C3 = (T0[2] * T1[2]) + (C2 * (T0[2] + T1[2]));
C4 = (T0[3] * T1[3]) + (C3 * (T0[3] + T1[3]));
OCOUT[0] = C4;
O0[0] = ICIN[0] ^ T0[0] ^ T1[0];
O0[1] = C1 ^ T0[1] ^ T1[1];
O0[2] = C2 ^ T0[2] ^ T1[2];
O0[3] = C3 ^ T0[3] ^ T1[3];

OVF[0] = C3 ^ C4;
```

F.2 ALU

```
------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                   GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (GC_ADD, GC_LNOT) GC_STYLE = GC_CARRY_LOOKAHEAD
------------------------------------------------------------------
INORDER = CADD[0] CLNOT[0] I0[0] I0[1] I0[2] I0[3] I1[0]
          I1[1] I1[2] I1[3] ICIN[0];
OUTORDER = O0[0] O0[1] O0[2] O0[3] OCOUT[0] O1[0] OZERO[0] OVF[0];

C0 = (CADD[0]*ICIN[0]);
X[0] = (CADD[0]*I0[0]);
X[1] = (CADD[0]*I0[1]);
X[2] = (CADD[0]*I0[2]);
X[3] = (CADD[0]*I0[3]);
Y[0] = (CADD[0]*I1[0]);
Y[1] = (CADD[0]*I1[1]);
Y[2] = (CADD[0]*I1[2]);
Y[3] = (CADD[0]*I1[3]);

AB[0]   = I0[0]  * I1[0];
ABN[0]  = I0[0]  * !I1[0];
ANB[0]  = !I0[0] * I1[0];
ANBN[0] = !I0[0] * !I1[0];
```

```
AB[1]   = I0[1]  * I1[1];
ABN[1]  = I0[1]  * !I1[1];
ANB[1]  = !I0[1] * I1[1];
ANBN[1] = !I0[1] * !I1[1];

AB[2]   = I0[2]  * I1[2];
ABN[2]  = I0[2]  * !I1[2];
ANB[2]  = !I0[2] * I1[2];
ANBN[2] = !I0[2] * !I1[2];

AB[3]   = I0[3]  * I1[3];
ABN[3]  = I0[3]  * !I1[3];
ANB[3]  = !I0[3] * I1[3];
ANBN[3] = !I0[3] * !I1[3];

ANBAND[0] = ANB[0] * CLNOT[0];
ANBNAND[0] = ANBN[0] * CLNOT[0];

ZL[0] = ANBAND[0] + ANBNAND[0];

ANBAND[1] = ANB[1] * CLNOT[0];
ANBNAND[1] = ANBN[1] * CLNOT[0];

ZL[1] = ANBAND[1] + ANBNAND[1];

ANBAND[2] = ANB[2] * CLNOT[0];
ANBNAND[2] = ANBN[2] * CLNOT[0];

ZL[2] = ANBAND[2] + ANBNAND[2];

ANBAND[3] = ANB[3] * CLNOT[0];
ANBNAND[3] = ANBN[3] * CLNOT[0];

ZL[3] = ANBAND[3] + ANBNAND[3];

G0[0] = X[0] * Y[0];
P0[0] = X[0] + Y[0];
G0[1] = X[1] * Y[1];
P0[1] = X[1] + Y[1];
G0[2] = X[2] * Y[2];
P0[2] = X[2] + Y[2];
G0[3] = X[3] * Y[3];
P0[3] = X[3] + Y[3];

P1[1] = P0[0] * P0[1];
G1[1] = P0[1] * G0[0] + G0[1];
P1[3] = P0[2] * P0[3];
G1[3] = P0[3] * G0[2] + G0[3];

P2[2] = P1[1] * P0[2];
G2[2] = P0[2] * G1[1] + G0[2];
P2[3] = P1[1] * P1[3];
G2[3] = P1[3] * G1[1] + G1[3];

C1 = G0[0] + P0[0] * C0;
C2 = G1[1] + P1[1] * C0;
C3 = G2[2] + P2[2] * C0;
C4 = G2[3] + P2[3] * C0;
```

```
Z[0] = C0 ^ X[0] ^ Y[0];
Z[1] = C1 ^ X[1] ^ Y[1];
Z[2] = C2 ^ X[2] ^ Y[2];
Z[3] = C3 ^ X[3] ^ Y[3];

O1[0] = (0);
OCOUT[0] = C4;
OVF[0] = C4 ^ C3;

MS = 0 + CADD[0];
Z2[0] = (ZL[0]*!MS) + (Z[0]*MS);
Z2[1] = (ZL[1]*!MS) + (Z[1]*MS);
Z2[2] = (ZL[2]*!MS) + (Z[2]*MS);
Z2[3] = (ZL[3]*!MS) + (Z[3]*MS);

O0[0] = Z2[0];
O0[1] = Z2[1];
O0[2] = Z2[2];
O0[3] = Z2[3];

OZERO[0] = !(Z2[0] + Z2[1] + Z2[2] + Z2[3]);
```

F.3 AND

```
--------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                  GC_NUM_INPUTS = 4
--------------------------------------------------------------------
INORDER = I0[0] I1[0] I2[0] I3[0] I0[1] I1[1] I2[1] I3[1] I0[2] I1[2]
I2[2] I3[2] I0[3] I1[3] I2[3] I3[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

O0[0] = I0[0] * I1[0] * I2[0] * I3[0];
O0[1] = I0[1] * I1[1] * I2[1] * I3[1];
O0[2] = I0[2] * I1[2] * I2[2] * I3[2];
O0[3] = I0[3] * I1[3] * I2[3] * I3[3];
```

F.4 BARRELSHIFTER

```
--------------------------------------------------------------
-- GC_INPUT_WIDTH = 5                     GC_NUM_FUNCTIONS = 4
-- GC_FUNCTION_LIST = (GC_SHR, GC_SHL, GC_ROTR, ROTL)
-- GC_SHIFT_DISTANCE = 5
--------------------------------------------------------------
INORDER = CSHR[0] CSHL[0] CROTR[0] CROTL[0] I0[4] I0[3] I0[2]
I0[1] I0[0] ISHNUM[2] ISHNUM[1] ISHNUM[0] IFILL[0];
OUTORDER = O0[4] O0[3] O0[2] O0[1] O0[0];

MUX[3][4]O = I0[4];
MUX[3][3]O = I0[3];
MUX[3][2]O = I0[2];
MUX[3][1]O = I0[1];
MUX[3][0]O = I0[0];

MUX[2][4]I0 = MUX[3][4]O;
MUX[2][4]I1 = (CSHL[0] * MUX[3][0]O) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[3][0]O) + (CROTR[0] * MUX[3][3]O) + 0;
```

```
MUX[2][4]O = !(ISHNUM[2]) * MUX[2][4]I0 + ISHNUM[2] * MUX[2][4]I1;

MUX[2][3]I0 = MUX[3][3]O;
MUX[2][3]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[3][4]O) + (CROTR[0] * MUX[3][2]O) + 0;
MUX[2][3]O = !(ISHNUM[2]) * MUX[2][3]I0 + ISHNUM[2] * MUX[2][3]I1;

MUX[2][2]I0 = MUX[3][2]O;
MUX[2][2]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[3][3]O) + (CROTR[0] * MUX[3][1]O) + 0;
MUX[2][2]O = !(ISHNUM[2]) * MUX[2][2]I0 + ISHNUM[2] * MUX[2][2]I1;

MUX[2][1]I0 = MUX[3][1]O;
MUX[2][1]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[3][2]O) + (CROTR[0] * MUX[3][0]O) + 0;
MUX[2][1]O = !(ISHNUM[2]) * MUX[2][1]I0 + ISHNUM[2] * MUX[2][1]I1;

MUX[2][0]I0 = MUX[3][0]O;
MUX[2][0]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * MUX[3][4]O) +
(CROTL[0] * MUX[3][1]O) + (CROTR[0] * MUX[3][4]O) + 0;
MUX[2][0]O = !(ISHNUM[2]) * MUX[2][0]I0 + ISHNUM[2] * MUX[2][0]I1;

MUX[1][4]I0 = MUX[2][4]O;
MUX[1][4]I1 = (CSHL[0] * MUX[2][2]O) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[2][2]O) + (CROTR[0] * MUX[2][1]O) + 0;
MUX[1][4]O = !(ISHNUM[1]) * MUX[1][4]I0 + ISHNUM[1] * MUX[1][4]I1;

MUX[1][3]I0 = MUX[2][3]O;
MUX[1][3]I1 = (CSHL[0] * MUX[2][1]O) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[2][1]O) + (CROTR[0] * MUX[2][0]O) + 0;
MUX[1][3]O = !(ISHNUM[1]) * MUX[1][3]I0 + ISHNUM[1] * MUX[1][3]I1;

MUX[1][2]I0 = MUX[2][2]O;
MUX[1][2]I1 = (CSHL[0] * MUX[2][0]O) + (CSHR[0] * MUX[2][4]O) +
(CROTL[0] * MUX[2][0]O) + (CROTR[0] * MUX[2][4]O) + 0;
MUX[1][2]O = !(ISHNUM[1]) * MUX[1][2]I0 + ISHNUM[1] * MUX[1][2]I1;

MUX[1][1]I0 = MUX[2][1]O;
MUX[1][1]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * MUX[2][3]O) +
(CROTL[0] * MUX[2][4]O) + (CROTR[0] * MUX[2][3]O) + 0;
MUX[1][1]O = !(ISHNUM[1]) * MUX[1][1]I0 + ISHNUM[1] * MUX[1][1]I1;

MUX[1][0]I0 = MUX[2][0]O;
MUX[1][0]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * MUX[2][2]O) +
(CROTL[0] * MUX[2][3]O) + (CROTR[0] * MUX[2][2]O) + 0;
MUX[1][0]O = !(ISHNUM[1]) * MUX[1][0]I0 + ISHNUM[1] * MUX[1][0]I1;

MUX[0][4]I0 = MUX[1][4]O;
MUX[0][4]I1 = (CSHL[0] * MUX[1][3]O) + (CSHR[0] * IFILL[0]) +
(CROTL[0] * MUX[1][3]O) + (CROTR[0] * MUX[1][0]O) + 0;
MUX[0][4]O = !(ISHNUM[0]) * MUX[0][4]I0 + ISHNUM[0] * MUX[0][4]I1;

MUX[0][3]I0 = MUX[1][3]O;
MUX[0][3]I1 = (CSHL[0] * MUX[1][2]O) + (CSHR[0] * MUX[1][4]O) +
(CROTL[0] * MUX[1][2]O) + (CROTR[0] * MUX[1][4]O) + 0;
MUX[0][3]O = !(ISHNUM[0]) * MUX[0][3]I0 + ISHNUM[0] * MUX[0][3]I1;

MUX[0][2]I0 = MUX[1][2]O;
MUX[0][2]I1 = (CSHL[0] * MUX[1][1]O) + (CSHR[0] * MUX[1][3]O) +
```

```
(CROTL[0] * MUX[1][1]O) + (CROTR[0] * MUX[1][3]O) + 0;
MUX[0][2]O = !(ISHNUM[0]) * MUX[0][2]I0 + ISHNUM[0] * MUX[0][2]I1;

MUX[0][1]I0 = MUX[1][1]O;
MUX[0][1]I1 = (CSHL[0] * MUX[1][0]O) + (CSHR[0] * MUX[1][2]O) +
(CROTL[0] * MUX[1][0]O) + (CROTR[0] * MUX[1][2]O) + 0;
MUX[0][1]O = !(ISHNUM[0]) * MUX[0][1]I0 + ISHNUM[0] * MUX[0][1]I1;

MUX[0][0]I0 = MUX[1][0]O;
MUX[0][0]I1 = (CSHL[0] * IFILL[0]) + (CSHR[0] * MUX[1][1]O) +
(CROTL[0] * MUX[1][4]O) + (CROTR[0] * MUX[1][1]O) + 0;
MUX[0][0]O = !(ISHNUM[0]) * MUX[0][0]I0 + ISHNUM[0] * MUX[0][0]I1;

OO[4] = MUX[0][4]O;
OO[3] = MUX[0][3]O;
OO[2] = MUX[0][2]O;
OO[1] = MUX[0][1]O;
OO[0] = MUX[0][0]O;
```

F.5 BUFFER

```
------------------------------------------------------
-- GC_INPUT_WIDTH = 4            GC_INVERT_FLAG = True
------------------------------------------------------
INORDER = IO[3] IO[2] IO[1] IO[0];
OUTORDER = OO[3] OO[2] OO[1] OO[0];

OO[3] = !IO[3];
OO[2] = !IO[2];
OO[1] = !IO[1];
OO[0] = !IO[0];
```

F.6 BUS

```
--------------------------------------------------------
-- GC_INPUT_WIDTH = 4             GC_NUM_INPUTS = 6
-- GC_FAN_OUT = 4
--------------------------------------------------------
INORDER = I0[0] I1[0] I2[0] I3[0] I4[0] I5[0] I0[1] I1[1]
I2[1] I3[1] I4[1] I5[1] I0[2] I1[2] I2[2] I3[2] I4[2] I5[2]
I0[3] I1[3] I2[3] I3[3] I4[3] I5[3] CI[0] CI[1] CI[2] CI[3]
CI[4] CI[5];
OUTORDER = OO[0] OO[1] OO[2] OO[3];

OO[0] = ((CI[0] * I0[0]) + (CI[1] * I1[0]) + (CI[2] * I2[0]) +
(CI[3] * I3[0]) + (CI[4] * I4[0]) + (CI[5] * I5[0]));
OO[1] = ((CI[0] * I0[1]) + (CI[1] * I1[1]) + (CI[2] * I2[1]) +
(CI[3] * I3[1]) + (CI[4] * I4[1]) + (CI[5] * I5[1]));
OO[2] = ((CI[0] * I0[2]) + (CI[1] * I1[2]) + (CI[2] * I2[2]) +
(CI[3] * I3[2]) + (CI[4] * I4[2]) + (CI[5] * I5[2]));
OO[3] = ((CI[0] * I0[3]) + (CI[1] * I1[3]) + (CI[2] * I2[3]) +
(CI[3] * I3[3]) + (CI[4] * I4[3]) + (CI[5] * I5[3]));
```

F.7 COMPARATOR

```
------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                  GC_NUM_FUNCTIONS = 3
```

```
-- GC_FUNCTION_LIST = (GC_NEQ, GC_LT, GC_LEQ)
--------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3] I1[0] I1[1] I1[2] I1[3] CNEQ[0]
CLT[0] CLEQ[0];
OUTORDER = ONEQ[0] OLT[0] OLEQ[0];

EQ[0] = (!(I0[0]) * !(I1[0])) + (I0[0] * I1[0]);
EQ[1] = (!(I0[1]) * !(I1[1])) + (I0[1] * I1[1]);
EQ[2] = (!(I0[2]) * !(I1[2])) + (I0[2] * I1[2]);
EQ[3] = (!(I0[3]) * !(I1[3])) + (I0[3] * I1[3]);

TEQ[0] = EQ[0] * EQ[1] * EQ[2] * EQ[3];

ONEQ[0] = CNEQ[0] * !(TEQ[0]);

LT[3] = !(I0[3]) * I1[3];
LT[2] = EQ[3] * !(I0[2]) * I1[2];
LT[1] = EQ[3] * EQ[2] * !(I0[1]) * I1[1];
LT[0] = EQ[3] * EQ[2] * EQ[1] * !(I0[0]) * I1[0];

TLT[0] = (LT[3] + LT[2] + LT[1] + LT[0]);

OLT[0] = CLT[0] * TLT[0];

GT[3] = I0[3] * !(I1[3]);
GT[2] = EQ[3] * I0[2] * !(I1[2]);
GT[1] = EQ[3] * EQ[2] * I0[1] * !(I1[1]);
GT[0] = EQ[3] * EQ[2] * EQ[1] * I0[0] * !(I1[0]);

TGT[0] = (GT[3] + GT[2] + GT[1] + GT[0]);

OLEQ[0] = CLEQ[0] * !(TGT[0]);
```

F.8 CONCAT

```
-----------------------------------------------------------------
-- GC_NUM_INPUTS = 4           GC_INPUT_WIDTH_LIST = (4, 3, 5, 2)
-----------------------------------------------------------------
INORDER = I0[3] I0[2] I0[1] I0[0] I1[2] I1[1] I1[0] I2[4] I2[3]
I2[2] I2[1] I2[0] I3[1] I3[0];
OUTORDER = O0[0] O0[1] O0[2] O0[3] O0[4] O0[5] O0[6] O0[7] O0[8]
O0[9] O0[10] O0[11] O0[12] O0[13];

O0[0] = I3[0];
O0[1] = I3[1];
O0[2] = I2[0];
O0[3] = I2[1];
O0[4] = I2[2];
O0[5] = I2[3];
O0[6] = I2[4];
O0[7] = I1[0];
O0[8] = I1[1];
O0[9] = I1[2];
O0[10] = I0[0];
O0[11] = I0[1];
O0[12] = I0[2];
O0[13] = I0[3];
```

F.9 CONSTANT

```
------------------------------------------------------
-- GC_OUTPUT_WIDTH = 4            GC_CONST_VALUE = 7
------------------------------------------------------
OUTORDER = O0[3] O0[2] O0[1] O0[0] ;

O0[0] = 1;
O0[1] = 1;
O0[2] = 1;
O0[3] = 0;
```

F.10 COUNTER

```
-----------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4             GC_NUM_FUNCTIONS = 3
-- GC_FUNCTION_LIST = (GC_LOAD, GC_COUNT_UP, GC_SCAN)
-- GC_TYPE = GC_BINARY            GC_STYLE = GC_RIPPLE
-- GC_SET_VALUE = 9
-----------------------------------------------------------------------
entity COUNTER_11M193Z2T3Z2 is
port (I0 : in BIT_VECTOR(3 downto 0);
      ISCAN : in BIT;
      CLK, ASET, ARESET, CLOAD, CUP, CSCAN : in BIT;
      O0 : out BIT_VECTOR(3 downto 0);
      OSCAN : out BIT);
end COUNTER_11M193Z2T3Z2;

-- LAYOUT_STYLE STANDARD_CELL

architecture COUNTER_11M193Z2T3Z2_behavior of COUNTER_11M193Z2T3Z2 is
   COMPONENT DFF
     PORT (CLOCK_2, ff_set, ff_reset, I: in bit;
           O0, O0N : out bit);
   END COMPONENT;

   COMPONENT zero
     PORT (O0 : out bit);
   END COMPONENT;

-- HIERARCHY COUNTER_11M193Z2T3Z2_logic1.eqn
   COMPONENT COUNTER_11M193Z2T3Z2_logic1
     PORT (L1_LOAD, L1_UP, L1_SCAN, ILOAD, IUP, ISCAN : in bit;
           O0 : out bit);
   END COMPONENT;

-- HIERARCHY COUNTER_11M193Z2T3Z2_logic3.eqn
   COMPONENT COUNTER_11M193Z2T3Z2_logic3
     PORT (I0, I1 : in bit;
           O0 : out bit);
   END COMPONENT;

-- HIERARCHY COUNTER_11M193Z2T3Z2_logic4.eqn
   COMPONENT COUNTER_11M193Z2T3Z2_logic4
     PORT (I0 : in bit;
           O0 : out bit);
     END COMPONENT;
```

```
SIGNAL D0, D1, D2, D3, SET0, RESL0 : bit;
SIGNAL TO0_0, TO0_1, TO0_2, TO0_3 : bit;
SIGNAL TO0N_0, TO0N_1, TO0N_2, TO0N_3 : bit;
SIGNAL CK_0, CK_1, CK_2, CK_3 : bit;
BEGIN  -- COUNTER_11M193Z2T3Z2
   FF0: DFF port map(CLOCK_1, ASET, ARESET, D0, TO0_0, TO0N_0);
   FF1: DFF port map(CK_1, SET0, RESL0, D1, TO0_1, TO0N_1);
   FF2: DFF port map(CK_2, SET0, RESL0, D2, TO0_2, TO0N_2);
   FF3: DFF port map(CK_3, ASET, ARESET, D3, TO0_3, TO0N_3);
   L1_0: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         I0_0, TO0N_0, TO0_1, D0);
   L1_1: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         I0_1, TO0N_1, TO0_2, D1);
   L1_2: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         I0_2, TO0N_2, TO0_3, D2);
   L1_3: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         I0_3, TO0N_3, ISCAN, D3);
   L1_5: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         CLOCK_1, TO0N_0, CLOCK_1, CK_1);
   L1_6: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         CLOCK_1, TO0N_1, CLOCK_1, CK_2);
   L1_7: COUNTER_11M193Z2T3Z2_logic1 port map(CLOAD, CUP, CSCAN,
         CLOCK_1, TO0N_2, CLOCK_1, CK_3);
   C0: zero port map(SET0);
   L3_0: COUNTER_11M193Z2T3Z2_logic3 port map(ASET, ARESET, RESL0);
   L4_1: COUNTER_11M193Z2T3Z2_logic4 port map(TO0_0, OSCAN);
   L4_2: COUNTER_11M193Z2T3Z2_logic4 port map(TO0_0,O0_0);
   L4_3: COUNTER_11M193Z2T3Z2_logic4 port map(TO0_1,O0_1);
   L4_4: COUNTER_11M193Z2T3Z2_logic4 port map(TO0_2,O0_2);
   L4_5: COUNTER_11M193Z2T3Z2_logic4 port map(TO0_3,O0_3);
END COUNTER_11M193Z2T3Z2;

-- COUNTER_11M193Z2T3Z2_logic1.eqn

INORDER =  CLOAD CUP CSCAN ILOAD IUP ISCAN;
OUTORDER = O0;

O0 = (CLOAD * ILOAD) + (CUP * IUP) + (CSCAN * ISCAN);

-- COUNTER_11M193Z2T3Z2_logic3.eqn

INORDER = I0 I1;
OUTORDER = O0;

O0 = I0 + I1;

-- COUNTER_11M193Z2T3Z2_logic4.eqn

INORDER = I0;
OUTORDER = O0;

O0 = I0;
```

F.11 DECODE

```
--------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
```

```
-----------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3] O0[4] O0[5] O0[6] O0[7] O0[8]
O0[9] O0[10] O0[11] O0[12] O0[13] O0[14] O0[15];

O0[0] = !I0[0] * !I0[1] * !I0[2] * !I0[3];
O0[1] =  I0[0] * !I0[1] * !I0[2] * !I0[3];
O0[2] = !I0[0] *  I0[1] * !I0[2] * !I0[3];
O0[3] =  I0[0] *  I0[1] * !I0[2] * !I0[3];
O0[4] = !I0[0] * !I0[1] *  I0[2] * !I0[3];
O0[5] =  I0[0] * !I0[1] *  I0[2] * !I0[3];
O0[6] = !I0[0] *  I0[1] *  I0[2] * !I0[3];
O0[7] =  I0[0] *  I0[1] *  I0[2] * !I0[3];
O0[8] = !I0[0] * !I0[1] * !I0[2] *  I0[3];
O0[9] =  I0[0] * !I0[1] * !I0[2] *  I0[3];
O0[10] = !I0[0] *  I0[1] * !I0[2] *  I0[3];
O0[11] =  I0[0] *  I0[1] * !I0[2] *  I0[3];
O0[12] = !I0[0] * !I0[1] *  I0[2] *  I0[3];
O0[13] =  I0[0] * !I0[1] *  I0[2] *  I0[3];
O0[14] = !I0[0] *  I0[1] *  I0[2] *  I0[3];
O0[15] =  I0[0] *  I0[1] *  I0[2] *  I0[3];
```

F.12 ENCODE

```
-----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
-----------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0] O0[1];

O0[0] = I0[1] + I0[3];
O0[1] = I0[2] + I0[3];
```

F.13 EXTRACT

```
-----------------------------------------------------------------
-- GC_INPUT_WIDTH = 10                      GC_LEFT_INDEX = 8
-- GC_RIGHT_INDEX = 2
-----------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3] I0[4] I0[5] I0[6] I0[7] I0[8] I0[9];
OUTORDER = O0[0] O0[1] O0[2] O0[3] O0[4] O0[5] O0[6];

O0[0] = I0[2];
O0[1] = I0[3];
O0[2] = I0[4];
O0[3] = I0[5];
O0[4] = I0[6];
O0[5] = I0[7];
O0[6] = I0[8];
```

F.14 GAND

```
-----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
-----------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];
```

```
O0[0] = I0[0] * I0[1] * I0[2] * I0[3];
```

F.15 GNAND

```
----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
----------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];

O0[0] = !(I0[0] * I0[1] * I0[2] * I0[3]);
```

F.16 GNOR

```
-----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
-----------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];

O0[0] = !(I0[0] + I0[1] + I0[2] + I0[3]);
```

F.17 GNOT

```
------------------------------------------------------------
------------------------------------------------------------
INORDER = I0[0];
OUTORDER = O0[0];

O0[0] = !(I0[0]);
```

F.18 GOR

```
------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];

O0[0] = I0[0] + I0[1] + I0[2] + I0[3];
```

F.19 GXNOR

```
---------------------------------------------------------------
-- GC_INPUT_WIDTH = 4
---------------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];

O0[0] = I0[0] == I0[1] == I0[2] == I0[3];
```

F.20 GXOR

```
--------------------------------------------------------
-- GC_INPUT_WIDTH = 4
--------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];

O0[0] = I0[0] != I0[1] != I0[2] != I0[3];
```

F.21 LU

```
--------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                    GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (GC_ZERO, GC_RNOT)
--------------------------------------------------------------
INORDER = CZERO[0] CRNOT[0] I0[0] I0[1] I0[2] I0[3] I1[0] I1[1] I1[2] I1[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

AB[0]   = I0[0]  * I1[0];
ABN[0]  = I0[0]  * !I1[0];
ANB[0]  = !I0[0] * I1[0];
ANBN[0] = !I0[0] * !I1[0];

AB[1]   = I0[1]  * I1[1];
ABN[1]  = I0[1]  * !I1[1];
ANB[1]  = !I0[1] * I1[1];
ANBN[1] = !I0[1] * !I1[1];

AB[2]   = I0[2]  * I1[2];
ABN[2]  = I0[2]  * !I1[2];
ANB[2]  = !I0[2] * I1[2];
ANBN[2] = !I0[2] * !I1[2];

AB[3]   = I0[3]  * I1[3];
ABN[3]  = I0[3]  * !I1[3];
ANB[3]  = !I0[3] * I1[3];
ANBN[3] = !I0[3] * !I1[3];

ABNAND[0] = ABN[0] * CRNOT[0];
ANBNAND[0] = ANBN[0] * CRNOT[0];

O0[0] = ABNAND[0] + ANBNAND[0];

ABNAND[1] = ABN[1] * CRNOT[0];
ANBNAND[1] = ANBN[1] * CRNOT[0];

O0[1] = ABNAND[1] + ANBNAND[1];

ABNAND[2] = ABN[2] * CRNOT[0];
ANBNAND[2] = ANBN[2] * CRNOT[0];

O0[2] = ABNAND[2] + ANBNAND[2];

ABNAND[3] = ABN[3] * CRNOT[0];
ANBNAND[3] = ANBN[3] * CRNOT[0];

O0[3] = ABNAND[3] + ANBNAND[3];
```

F.22 MULT

```
----------------------------------------------------------------
-- GC_LEFT_INPUT_WIDTH = 4          GC_RIGHT_INPUT_WIDTH = 3
-- GC_STYLE = GC_ARRAY              GC_PIPELINE_FLAG = False
-- GC_PIPELINE_STAGES = 4
----------------------------------------------------------------
INORDER = I0[3] I0[2] I0[1] I0[0] I1[2] I1[1] I1[0] ;
OUTORDER = O0[6] O0[5] O0[4] O0[3] O0[2] O0[1] O0[0] ;

CTRL[0]H = I1[2] ^ I1[1];
CTRL[0]D = I1[2] * !(I1[1]);
CTRL[1]H = I1[1] ^ I1[0];
CTRL[1]D = I1[1] * !(I1[0]);
CTRL[2]H = I1[0];
CTRL[2]D = I1[0];

CASS[-1][-2]A = I0[3];
CASS[-1][-1]P = 0;
CASS[-1][-1]A = I0[3];
CASS[-1][0]P = 0;
CASS[-1][0]A = I0[2];
CASS[-1][1]P = 0;
CASS[-1][1]A = I0[1];
CASS[-1][2]P = 0;
CASS[-1][2]A = I0[0];
CASS[-1][3]P = 0;

CASS[0][-2]H = CTRL[0]H;
CASS[0][-2]D = CTRL[0]D;
CASS[0][4]C = 0;
CASS[1][-2]H = CTRL[1]H;
CASS[1][-2]D = CTRL[1]D;
CASS[1][5]C = 0;
CASS[0][4]P = 0;
CASS[2][-2]H = CTRL[2]H;
CASS[2][-2]D = CTRL[2]D;
CASS[2][6]C = 0;
CASS[1][5]P = 0;

CASS[0][-1]H = CASS[0][-2]H;
CASS[0][-1]D = CASS[0][-2]D;
CASS[0][-1]A = CASS[-1][-2]A;
CASS[0][-1]P = CASS[-1][-1]P ^ (CASS[-1][-2]A * CASS[0][-2]H) ^
(CASS[0][0]C * CASS[0][-2]H);
CASS[0][-1]C = (CASS[-1][-1]P ^ CASS[0][-2]D) * (CASS[-1][-2]A +
CASS[0][0]C) + (CASS[-1][-2]A * CASS[0][0]C);

CASS[0][0]H = CASS[0][-1]H;
CASS[0][0]D = CASS[0][-1]D;
CASS[0][0]A = CASS[-1][-1]A;
CASS[0][0]P = CASS[-1][0]P ^ (CASS[-1][-1]A * CASS[0][-1]H) ^
(CASS[0][1]C * CASS[0][-1]H);
CASS[0][0]C = (CASS[-1][0]P ^ CASS[0][-1]D) * (CASS[-1][-1]A +
CASS[0][1]C) + (CASS[-1][-1]A * CASS[0][1]C);

CASS[0][1]H = CASS[0][0]H;
CASS[0][1]D = CASS[0][0]D;
CASS[0][1]A = CASS[-1][0]A;
```

```
CASS[0][1]P = CASS[-1][1]P ^ (CASS[-1][0]A * CASS[0][0]H) ^
(CASS[0][2]C * CASS[0][0]H);
CASS[0][1]C = (CASS[-1][1]P ^ CASS[0][0]D) * (CASS[-1][0]A +
CASS[0][2]C) + (CASS[-1][0]A * CASS[0][2]C);

CASS[0][2]H = CASS[0][1]H;
CASS[0][2]D = CASS[0][1]D;
CASS[0][2]A = CASS[-1][1]A;
CASS[0][2]P = CASS[-1][2]P ^ (CASS[-1][1]A * CASS[0][1]H) ^
(CASS[0][3]C * CASS[0][1]H);
CASS[0][2]C = (CASS[-1][2]P ^ CASS[0][1]D) * (CASS[-1][1]A +
CASS[0][3]C) + (CASS[-1][1]A * CASS[0][3]C);

CASS[0][3]H = CASS[0][2]H;
CASS[0][3]D = CASS[0][2]D;
CASS[0][3]A = CASS[-1][2]A;
CASS[0][3]P = CASS[-1][3]P ^ (CASS[-1][2]A * CASS[0][2]H) ^
(CASS[0][4]C * CASS[0][2]H);
CASS[0][3]C = (CASS[-1][3]P ^ CASS[0][2]D) * (CASS[-1][2]A +
CASS[0][4]C) + (CASS[-1][2]A * CASS[0][4]C);

CASS[0][-2]A = CASS[0][-1]A;

CASS[1][-1]H = CASS[1][-2]H;
CASS[1][-1]D = CASS[1][-2]D;
CASS[1][-1]A = CASS[0][-2]A;
CASS[1][-1]P = CASS[0][-1]P ^ (CASS[0][-2]A * CASS[1][-2]H) ^
(CASS[1][0]C * CASS[1][-2]H);
CASS[1][-1]C = (CASS[0][-1]P ^ CASS[1][-2]D) * (CASS[0][-2]A +
CASS[1][0]C) + (CASS[0][-2]A * CASS[1][0]C);

CASS[1][0]H = CASS[1][-1]H;
CASS[1][0]D = CASS[1][-1]D;
CASS[1][0]A = CASS[0][-1]A;
CASS[1][0]P = CASS[0][0]P ^ (CASS[0][-1]A * CASS[1][-1]H) ^
(CASS[1][1]C * CASS[1][-1]H);
CASS[1][0]C = (CASS[0][0]P ^ CASS[1][-1]D) * (CASS[0][-1]A +
CASS[1][1]C) + (CASS[0][-1]A * CASS[1][1]C);

CASS[1][1]H = CASS[1][0]H;
CASS[1][1]D = CASS[1][0]D;
CASS[1][1]A = CASS[0][0]A;
CASS[1][1]P = CASS[0][1]P ^ (CASS[0][0]A * CASS[1][0]H) ^
(CASS[1][2]C * CASS[1][0]H);
CASS[1][1]C = (CASS[0][1]P ^ CASS[1][0]D) * (CASS[0][0]A +
CASS[1][2]C) + (CASS[0][0]A * CASS[1][2]C);

CASS[1][2]H = CASS[1][1]H;
CASS[1][2]D = CASS[1][1]D;
CASS[1][2]A = CASS[0][1]A;
CASS[1][2]P = CASS[0][2]P ^ (CASS[0][1]A * CASS[1][1]H) ^
(CASS[1][3]C * CASS[1][1]H);
CASS[1][2]C = (CASS[0][2]P ^ CASS[1][1]D) * (CASS[0][1]A +
CASS[1][3]C) + (CASS[0][1]A * CASS[1][3]C);

CASS[1][3]H = CASS[1][2]H;
CASS[1][3]D = CASS[1][2]D;
CASS[1][3]A = CASS[0][2]A;
CASS[1][3]P = CASS[0][3]P ^ (CASS[0][2]A * CASS[1][2]H) ^
```

```
(CASS[1][4]C * CASS[1][2]H);
CASS[1][3]C = (CASS[0][3]P ^ CASS[1][2]D) * (CASS[0][2]A +
CASS[1][4]C) + (CASS[0][2]A * CASS[1][4]C);

CASS[1][4]H = CASS[1][3]H;
CASS[1][4]D = CASS[1][3]D;
CASS[1][4]A = CASS[0][3]A;
CASS[1][4]P = CASS[0][4]P ^ (CASS[0][3]A * CASS[1][3]H) ^
(CASS[1][5]C * CASS[1][3]H);
CASS[1][4]C = (CASS[0][4]P ^ CASS[1][3]D) * (CASS[0][3]A +
CASS[1][5]C) + (CASS[0][3]A * CASS[1][5]C);

CASS[1][-2]A = CASS[1][-1]A;

CASS[2][-1]H = CASS[2][-2]H;
CASS[2][-1]D = CASS[2][-2]D;
CASS[2][-1]A = CASS[1][-2]A;
CASS[2][-1]P = CASS[1][-1]P ^ (CASS[1][-2]A * CASS[2][-2]H) ^
(CASS[2][0]C * CASS[2][-2]H);
CASS[2][-1]C = (CASS[1][-1]P ^ CASS[2][-2]D) * (CASS[1][-2]A +
CASS[2][0]C) + (CASS[1][-2]A * CASS[2][0]C);

CASS[2][0]H = CASS[2][-1]H;
CASS[2][0]D = CASS[2][-1]D;
CASS[2][0]A = CASS[1][-1]A;
CASS[2][0]P = CASS[1][0]P ^ (CASS[1][-1]A * CASS[2][-1]H) ^
(CASS[2][1]C * CASS[2][-1]H);
CASS[2][0]C = (CASS[1][0]P ^ CASS[2][-1]D) * (CASS[1][-1]A +
CASS[2][1]C) + (CASS[1][-1]A * CASS[2][1]C);

CASS[2][1]H = CASS[2][0]H;
CASS[2][1]D = CASS[2][0]D;
CASS[2][1]A = CASS[1][0]A;
CASS[2][1]P = CASS[1][1]P ^ (CASS[1][0]A * CASS[2][0]H) ^
(CASS[2][2]C * CASS[2][0]H);
CASS[2][1]C = (CASS[1][1]P ^ CASS[2][0]D) * (CASS[1][0]A +
CASS[2][2]C) + (CASS[1][0]A * CASS[2][2]C);

CASS[2][2]H = CASS[2][1]H;
CASS[2][2]D = CASS[2][1]D;
CASS[2][2]A = CASS[1][1]A;
CASS[2][2]P = CASS[1][2]P ^ (CASS[1][1]A * CASS[2][1]H) ^
(CASS[2][3]C * CASS[2][1]H);
CASS[2][2]C = (CASS[1][2]P ^ CASS[2][1]D) * (CASS[1][1]A +
CASS[2][3]C) + (CASS[1][1]A * CASS[2][3]C);

CASS[2][3]H = CASS[2][2]H;
CASS[2][3]D = CASS[2][2]D;
CASS[2][3]A = CASS[1][2]A;
CASS[2][3]P = CASS[1][3]P ^ (CASS[1][2]A * CASS[2][2]H) ^
(CASS[2][4]C * CASS[2][2]H);
CASS[2][3]C = (CASS[1][3]P ^ CASS[2][2]D) * (CASS[1][2]A +
CASS[2][4]C) + (CASS[1][2]A * CASS[2][4]C);

CASS[2][4]H = CASS[2][3]H;
CASS[2][4]D = CASS[2][3]D;
CASS[2][4]A = CASS[1][3]A;
CASS[2][4]P = CASS[1][4]P ^ (CASS[1][3]A * CASS[2][3]H) ^
(CASS[2][5]C * CASS[2][3]H);
```

```
CASS[2][4]C = (CASS[1][4]P ^ CASS[2][3]D) * (CASS[1][3]A +
CASS[2][5]C) + (CASS[1][3]A * CASS[2][5]C);

CASS[2][5]H = CASS[2][4]H;
CASS[2][5]D = CASS[2][4]D;
CASS[2][5]A = CASS[1][4]A;
CASS[2][5]P = CASS[1][5]P ^ (CASS[1][4]A * CASS[2][4]H) ^
(CASS[2][6]C * CASS[2][4]H);
CASS[2][5]C = (CASS[1][5]P ^ CASS[2][4]D) * (CASS[1][4]A +
CASS[2][6]C) + (CASS[1][4]A * CASS[2][6]C);

CASS[2][-2]A = CASS[2][-1]A;

O0[6] = CASS[2][-1]P;
O0[5] = CASS[2][0]P;
O0[4] = CASS[2][1]P;
O0[3] = CASS[2][2]P;
O0[2] = CASS[2][3]P;
O0[1] = CASS[2][4]P;
O0[0] = CASS[2][5]P;
```

F.23 MUX

```
--------------------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4            GC_NUM_INPUTS = 4
--------------------------------------------------------------------------------
INORDER = I0[0] I1[0] I2[0] I3[0] I0[1] I1[1] I2[1] I3[1] I0[2] I1[2] I2[2] I3[2]
I0[3] I1[3] I2[3] I3[3] CI[0] CI[1] CI[2] CI[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

O0[0] = ((CI[0] * I0[0]) + (CI[1] * I1[0]) + (CI[2] * I2[0]) + (CI[3] * I3[0]));
O0[1] = ((CI[0] * I0[1]) + (CI[1] * I1[1]) + (CI[2] * I2[1]) + (CI[3] * I3[1]));
O0[2] = ((CI[0] * I0[2]) + (CI[1] * I1[2]) + (CI[2] * I2[2]) + (CI[3] * I3[2]));
O0[3] = ((CI[0] * I0[3]) + (CI[1] * I1[3]) + (CI[2] * I2[3]) + (CI[3] * I3[3]));
```

F.24 NAND

```
----------------------------------------------------------
-- GC_INPUT_WIDTH = 4          GC_NUM_INPUTS = 4
----------------------------------------------------------
INORDER = I0[0] I1[0] I2[0] I3[0] I0[1] I1[1] I2[1] I3[1]
I0[2] I1[2] I2[2] I3[2] I0[3] I1[3] I2[3] I3[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

O0[0] = !(I0[0] * I1[0] * I2[0] * I3[0]);
O0[1] = !(I0[1] * I1[1] * I2[1] * I3[1]);
O0[2] = !(I0[2] * I1[2] * I2[2] * I3[2]);
O0[3] = !(I0[3] * I1[3] * I2[3] * I3[3]);
```

F.25 NOR

```
---------------------------------------------------------
-- GC_INPUT_WIDTH = 4          GC_NUM_INPUTS = 4
---------------------------------------------------------
INORDER = I0[0] I1[0] I2[0] I3[0] I0[1] I1[1] I2[1] I3[1]
I0[2] I1[2] I2[2] I3[2] I0[3] I1[3] I2[3] I3[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];
```

```
O0[0] = !(I0[0] + I1[0] + I2[0] + I3[0]);
O0[1] = !(I0[1] + I1[1] + I2[1] + I3[1]);
O0[2] = !(I0[2] + I1[2] + I2[2] + I3[2]);
O0[3] = !(I0[3] + I1[3] + I2[3] + I3[3]);
```

F.26 OR

```
----------------------------------------------------------
-- GC_INPUT_WIDTH = 4           GC_NUM_INPUTS = 2
----------------------------------------------------------
INORDER = I0[0] I1[0] I0[1] I1[1] I0[2] I1[2] I0[3] I1[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

O0[0] = I0[0] + I1[0];
O0[1] = I0[1] + I1[1];
O0[2] = I0[2] + I1[2];
O0[3] = I0[3] + I1[3];
```

F.27 PARITY

```
-------------------------------------------------------
-- GC_INPUT_WIDTH = 4            GC_TYPE = GC_EVEN
-------------------------------------------------------
INORDER = I0[0] I0[1] I0[2] I0[3];
OUTORDER = O0[0];

O0[0] = !( I0[0] ^ I0[1] ^ I0[2] ^ I0[3]);
```

F.28 PORT

```
--------------------------------------------------
-- GC_INPUT_WIDTH = 4       GC_MODE = GC_INPUT
--------------------------------------------------
INORDER = O0[3] O0[2] O0[1] O0[0];
OUTORDER = I0[3] I0[2] I0[1] I0[0];

I0[3] = O0[3];
I0[2] = O0[2];
I0[1] = O0[1];
I0[0] = O0[0];
```

F.29 REGFILE

```
----------------------------------------------------------------
-- GC_INPUT_WIDTH = 3               GC_NUM_INPUT_PORTS = 1
-- GC_NUM_OUTPUT_PORTS = 2          GC_NUM_INOUT_PORTS = 0
-- GC_NUM_WORDS = 4
----------------------------------------------------------------
entity REG_FILE_W31rwo is
port (I0_0, I0_1, I0_2 : in bit;
      IW0_0, IW0_1 : in bit;
      IR0_0, IR0_1 : in bit;
      IR1_0, IR1_1 : in bit;
      CW0 : in bit;
      CR0, CR1 : out bit;
```

```
      OO_0, OO_1, OO_2 : in bit;
      OO_0, OO_1, OO_2 : in bit
);
end REG_FILE_W31rwo;

-- LAYOUT_STYLE STANDARD_CELL

architecture REG_FILE_W31rwo_behavior of REG_FILE_W31rwo is
   COMPONENT DFF
     PORT (CLOCK_2, I: in bit;
           OO : out bit);
   END COMPONENT;

-- HIERARCHY REG_FILE_W31rwo_logic1.eqn
   COMPONENT logic1
     PORT (I00, I01, I02, CNONE, INONE : in bit;
           OO : out bit);
   END COMPONENT;

-- HIERARCHY REG_FILE_W31rwo_logic2.eqn
   COMPONENT logic2
     PORT (I00, I01 : in bit;
           OO : out bit);
   END COMPONENT;

-- HIERARCHY REG_FILE_W31rwo_logic3.eqn
   COMPONENT logic3
     PORT (I00, I01, I02, I10, I11, I12, I20, I21, I22, I30, I31, I32 : in bit;
           OO : out bit);
   END COMPONENT;

-- HIERARCHY REG_FILE_W31rwo_logic6.eqn
   COMPONENT logic6
     PORT (I0, I1 : in bit;
           O0, O1, O2, O3 : out bit);
     END COMPONENT;


SIGNAL D_0_0, D_0_1, D_0_2 : bit;
SIGNAL D_1_0, D_1_1, D_1_2 : bit;
SIGNAL D_2_0, D_2_1, D_2_2 : bit;
SIGNAL D_3_0, D_3_1, D_3_2 : bit;
SIGNAL T_0_0, T_0_1, T_0_2 : bit;
SIGNAL T_1_0, T_1_1, T_1_2 : bit;
SIGNAL T_2_0, T_2_1, T_2_2 : bit;
SIGNAL T_3_0, T_3_1, T_3_2 : bit;
SIGNAL IW_0_0, IW_0_1, IW_0_2, IW_0_3 : bit;
SIGNAL IR_0_0, IR_0_1, IR_0_2, IR_0_3 : bit;
SIGNAL IR_1_0, IR_1_1, IR_1_2, IR_1_3 : bit;
SIGNAL CNONE_0_0, CNONE_0_1, CNONE_0_2 : bit;
SIGNAL CNONE_1_0, CNONE_1_1, CNONE_1_2 : bit;
SIGNAL CNONE_2_0, CNONE_2_1, CNONE_2_2 : bit;
SIGNAL CNONE_3_0, CNONE_3_1, CNONE_3_2 : bit;

BEGIN  -- REG_FILE_W31rwo
   FF_0_0: DFF port map(CLOCK_1, D_0_0, T_0_0);
   FF_0_1: DFF port map(CLOCK_1, D_0_1, T_0_1);
   FF_0_2: DFF port map(CLOCK_1, D_0_2, T_0_2);
   FF_1_0: DFF port map(CLOCK_1, D_1_0, T_1_0);
```

```
   FF_1_1: DFF port map(CLOCK_1, D_1_1, T_1_1);
   FF_1_2: DFF port map(CLOCK_1, D_1_2, T_1_2);
   FF_2_0: DFF port map(CLOCK_1, D_2_0, T_2_0);
   FF_2_1: DFF port map(CLOCK_1, D_2_1, T_2_1);
   FF_2_2: DFF port map(CLOCK_1, D_2_2, T_2_2);
   FF_3_0: DFF port map(CLOCK_1, D_3_0, T_3_0);
   FF_3_1: DFF port map(CLOCK_1, D_3_1, T_3_1);
   FF_3_2: DFF port map(CLOCK_1, D_3_2, T_3_2);
   L1_0_0: logic1 port map(CW0, I0_0, IW_0_0, CNONE_0_0, T_0_0, D_0_0);
   L1_0_1: logic1 port map(CW0, I0_1, IW_0_0, CNONE_0_1, T_0_1, D_0_1);
   L1_0_2: logic1 port map(CW0, I0_2, IW_0_0, CNONE_0_2, T_0_2, D_0_2);
   L1_1_0: logic1 port map(CW0, I0_0, IW_0_1, CNONE_1_0, T_1_0, D_1_0);
   L1_1_1: logic1 port map(CW0, I0_1, IW_0_1, CNONE_1_1, T_1_1, D_1_1);
   L1_1_2: logic1 port map(CW0, I0_2, IW_0_1, CNONE_1_2, T_1_2, D_1_2);
   L1_2_0: logic1 port map(CW0, I0_0, IW_0_2, CNONE_2_0, T_2_0, D_2_0);
   L1_2_1: logic1 port map(CW0, I0_1, IW_0_2, CNONE_2_1, T_2_1, D_2_1);
   L1_2_2: logic1 port map(CW0, I0_2, IW_0_2, CNONE_2_2, T_2_2, D_2_2);
   L1_3_0: logic1 port map(CW0, I0_0, IW_0_3, CNONE_3_0, T_3_0, D_3_0);
   L1_3_1: logic1 port map(CW0, I0_1, IW_0_3, CNONE_3_1, T_3_1, D_3_1);
   L1_3_2: logic1 port map(CW0, I0_2, IW_0_3, CNONE_3_2, T_3_2, D_3_2);
   L2_0_0: logic2 port map(CW0, IW_0_0, CNONE_0_0);
   L2_0_1: logic2 port map(CW0, IW_0_0, CNONE_0_1);
   L2_0_2: logic2 port map(CW0, IW_0_0, CNONE_0_2);
   L2_1_0: logic2 port map(CW0, IW_0_1, CNONE_1_0);
   L2_1_1: logic2 port map(CW0, IW_0_1, CNONE_1_1);
   L2_1_2: logic2 port map(CW0, IW_0_1, CNONE_1_2);
   L2_2_0: logic2 port map(CW0, IW_0_2, CNONE_2_0);
   L2_2_1: logic2 port map(CW0, IW_0_2, CNONE_2_1);
   L2_2_2: logic2 port map(CW0, IW_0_2, CNONE_2_2);
   L2_3_0: logic2 port map(CW0, IW_0_3, CNONE_3_0);
   L2_3_1: logic2 port map(CW0, IW_0_3, CNONE_3_1);
   L2_3_2: logic2 port map(CW0, IW_0_3, CNONE_3_2);
   L3_0_0: logic3 port map(CR0, IR_0_0, T_0_0, CR0, IR_0_1, T_1_0, CR0,
                           IR_0_2, T_2_0, CR0, IR_0_3, T_3_0, O0_0);
   L3_0_1: logic3 port map(CR0, IR_0_0, T_0_1, CR0, IR_0_1, T_1_1, CR0,
                           IR_0_2, T_2_1, CR0, IR_0_3, T_3_1, O0_1);
   L3_0_2: logic3 port map(CR0, IR_0_0, T_0_2, CR0, IR_0_1, T_1_2, CR0,
                           IR_0_2, T_2_2, CR0, IR_0_3, T_3_2, O0_2);
   L3_1_0: logic3 port map(CR1, IR_1_0, T_0_0, CR1, IR_1_1, T_1_0, CR1,
                           IR_1_2, T_2_0, CR1, IR_1_3, T_3_0, O1_0);
   L3_1_1: logic3 port map(CR1, IR_1_0, T_0_1, CR1, IR_1_1, T_1_1, CR1,
                           IR_1_2, T_2_1, CR1, IR_1_3, T_3_1, O1_1);
   L3_1_2: logic3 port map(CR1, IR_1_0, T_0_2, CR1, IR_1_1, T_1_2, CR1,
                           IR_1_2, T_2_2, CR1, IR_1_3, T_3_2, O1_2);
   L6_0_0: logic6 port map(IW0_0, IW0_1, IW_0_0, IW_0_1, IW_0_2, IW_0_3);
   L6_1_0: logic6 port map(IR0_0, IR0_1, IR_0_0, IR_0_1, IR_0_2, IR_0_3);
   L6_1_1: logic6 port map(IR1_0, IR1_1, IR_1_0, IR_1_1, IR_1_2, IR_1_3);
END REG_FILE_W31rwo;

-- REG_FILE_W31rwo_logic1.eqn

INORDER = I00 I01 I02 CNONE INONE;
OUTORDER = O0;

O0 = (CNONE * INONE) + (I00 * I01 * I02);

-- REG_FILE_W31rwo_logic2.eqn

INORDER = CI0 I0;
```

```
OUTORDER = O0;

O0 = !((CI0 * I0));

-- REG_FILE_W31rwo_logic3.eqn

INORDER = I00 I01 I02 I10 I11 I12 I20 I21 I22 I30 I31 I32 ;
OUTORDER = O0;

O0 = ((I00 * I01 * I02) + (I10 * I11 * I12) + (I20 * I21 * I22) +
     (I30 * I31 * I32));

-- REG_FILE_W31rwo_logic6.eqn

INORDER = I0 I1 ;
OUTORDER = O0 O1 O2 O3 ;

O0 = !I0 * !I1;
O1 = I0 * !I1;
O2 = !I0 * I1;
O3 = I0 * I1;
```

F.30 REGISTER

```
-----------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                 GC_NUM_FUNCTIONS = 3
-- GC_FUNCTION_LIST = (GC_LOAD, GC_SHR, GC_SCAN)
-- GC_TYPE = GC_D_FF                  GC_SET_VALUE = 10
-- GC_INVERT_OUT = True
-----------------------------------------------------------------
entity REGISTER_11M1B4835103 is
port (I0 : in BIT_VECTOR(3 downto 0);
      ILIN, ISCAN, CLK, ASET, ARESET, CLOAD, CSHR, CSCAN : in BIT;
      OON : out BIT_VECTOR(3 downto 0);
      OROUT, OSCAN : out BIT);
end REGISTER_11M1B4835103;

architecture REGISTER_11M1B4835103_behavior of REGISTER_11M1B4835103 is
   COMPONENT DFF
     PORT (CLK, ff_set, ff_reset, I: in bit;
           O0, O0N : out bit);
   END COMPONENT;

   COMPONENT zero
     PORT (O0 : out bit);
   END COMPONENT;

-- HIERARCHY REGISTER_11M1B4835103_logic1.eqn
   COMPONENT REGISTER_11M1B4835103_logic1
     PORT (L1_LOAD, L1_SHR, L1_SCAN, L1_NONE, ILOAD, ISHR, ISCAN, INONE : in bit;
           O0 : out bit);
   END COMPONENT;

-- HIERARCHY REGISTER_11M1B4835103_logic2.eqn
   COMPONENT REGISTER_11M1B4835103_logic2
     PORT (I0,I1,I2 : in bit;
           O0 : out bit);
   END COMPONENT;
```

```
-- HIERARCHY REGISTER_11M1B4835103_logic3.eqn
   COMPONENT REGISTER_11M1B4835103_logic3
     PORT (I0, I1 : in bit;
           O0 : out bit);
   END COMPONENT;

-- HIERARCHY REGISTER_11M1B4835103_logic4.eqn
   COMPONENT REGISTER_11M1B4835103_logic4
     PORT (I0 : in bit;
           O0 : out bit);
     END COMPONENT;


SIGNAL D0, D1, D2, D3, SET0, RESLO, CNONE : bit;
SIGNAL TO0_0, TO0_1, TO0_2, TO0_3 : bit;
SIGNAL TO0N_0, TO0N_1, TO0N_2, TO0N_3 : bit;

BEGIN  -- REGISTER_11M1B4835103
   FF0: DFF port map(CLK, SET0, RESLO, D0, TO0_0, TO0N_0);
   FF1: DFF port map(CLK, ASET, ARESET, D1, TO0_1, TO0N_1);
   FF2: DFF port map(CLK, SET0, RESLO, D2, TO0_2, TO0N_2);
   FF3: DFF port map(CLK, ASET, ARESET, D3, TO0_3, TO0N_3);
   L1_0: REGISTER_11M1B4835103_logic1 port map(CLOAD, CSHR, CSCAN,
         CNONE, I0(0), TO0_1, TO0_1, TO0_0, D0);
   L1_1: REGISTER_11M1B4835103_logic1 port map(CLOAD, CSHR, CSCAN,
         CNONE, I0(1), TO0_2, TO0_2, TO0_1, D1);
   L1_2: REGISTER_11M1B4835103_logic1 port map(CLOAD, CSHR, CSCAN,
         CNONE, I0(2), TO0_3, TO0_3, TO0_2, D2);
   L1_3: REGISTER_11M1B4835103_logic1 port map(CLOAD, CSHR, CSCAN,
         CNONE, I0(3), ILIN, ISCAN, TO0_3, D3);
   L2_0: REGISTER_11M1B4835103_logic2 port map(CLOAD, CSHR, CSCAN, CNONE);
   C0: zero port map(SET0);
   L3_0: REGISTER_11M1B4835103_logic3 port map(ASET, ARESET, RESLO);
   L4_1: REGISTER_11M1B4835103_logic4 port map(TO0_0, OROUT);
   L4_1: REGISTER_11M1B4835103_logic4 port map(TO0_0, OSCAN);
   L4_2: REGISTER_11M1B4835103_logic4 port map(TO0N_0,O0N(0));
   L4_3: REGISTER_11M1B4835103_logic4 port map(TO0N_1,O0N(1));
   L4_4: REGISTER_11M1B4835103_logic4 port map(TO0N_2,O0N(2));
   L4_5: REGISTER_11M1B4835103_logic4 port map(TO0N_3,O0N(3));
END REGISTER_11M1B4835103;

-- REGISTER_11M1B4835103_logic1.eqn

INORDER =  CLOAD CSHR CSCAN CNONE ILOAD ISHR ISCAN INONE;
OUTORDER = O0;

O0 = (CNONE * INONE) + (CLOAD * ILOAD) + (CSHR * ISHR) + (CSCAN * ISCAN);

-- REGISTER_11M1B4835103_logic2.eqn

INORDER = I0 I1 I2;
OUTORDER = O0;

O0 = !(I0 + I1 + I2);

-- REGISTER_11M1B4835103_logic3.eqn

INORDER = I0 I1;
```

```
OUTORDER = O0;

O0 = I0 + I1;

-- REGISTER_11M1B4835103_logic4.eqn

INORDER = I0;
OUTORDER = O0;

O0 = I0;
```

F.31 SHIFTER

```
-------------------------------------------------------------------
-- GC_INPUT_WIDTH = 4                   GC_NUM_FUNCTIONS = 2
-- GC_FUNCTION_LIST = (GC_SHL, GC_ROTL)
-------------------------------------------------------------------
INORDER = CSHL[0] CROTL[0] I0[3] I0[2] I0[1] I0[0] ILIN[0] IRIN[0];
OUTORDER = O0[3] O0[2] O0[1] O0[0] OROUT[0] OLOUT[0];

O0[0] = CSHL[0] * IRIN[0] + CROTL[0] * I0[3] + 0;
O0[1] = CSHL[0] * I0[0] + CROTL[0] * I0[0] + 0;
O0[2] = CSHL[0] * I0[1] + CROTL[0] * I0[1] + 0;
O0[3] = CSHL[0] * I0[2] + CROTL[0] * I0[2] + 0;
OLOUT[0] = I0[3];
OROUT[0] = 0;
```

F.32 XNOR

```
----------------------------------------------------------
-- GC_INPUT_WIDTH = 4           GC_NUM_INPUTS = 2
---------------------------------------------------------
INORDER = I0[0] I1[0] I0[1] I1[1] I0[2] I1[2] I0[3] I1[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

O0[0] = I0[0] == I1[0];
O0[1] = I0[1] == I1[1];
O0[2] = I0[2] == I1[2];
O0[3] = I0[3] == I1[3];
```

F.33 XOR

```
----------------------------------------------------------
-- GC_INPUT_WIDTH = 4                 GC_NUM_INPUTS = 2
----------------------------------------------------------
INORDER = I0[0] I1[0] I0[1] I1[1] I0[2] I1[2] I0[3] I1[3];
OUTORDER = O0[0] O0[1] O0[2] O0[3];

O0[0] = I0[0] != I1[0];
O0[1] = I0[1] != I1[1];
O0[2] = I0[2] != I1[2];
O0[3] = I0[3] != I1[3];
```

Index