The BIF User Interface and Programming Manual

Tedd Hadley

Technical Report #90-07
July 27, 1990

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-7063

hadley@ics.uci.edu

Abstract

This report is in two parts. The first part describes XBIF, the graphical/tabular editor for BIF (Behavioral Intermediate Format). The second part describes a library of routines that make up a programming interface to the BIF language.

Contents

List of Figures

1 XBIF

XBIF is a graphical/tabular interface to the Behavioral Intermediate Format (BIF). (BIF is described in detail in [DuHG89] and [DuHG90].) XBIF exploits the tabular nature of BIF by providing labeled forms to capture the various fields of a BIF table. XBIF performs syntax checking at runtime and provides visual feedback as to the location of errors.

XBIF operates directly and solely on syntactically correct BIF textual files.

XBIF is written under the X11 Window System using the Athena widget libary and the Xcu widget set from Cornell University.

1.1 The Top Level Window

XBIF is started initially with the command *xbif* [*BIF file*]. Figure 1 shows a screen dump of a sample XBIF description at the top level. The window lists all of the BIF tables in the design description in the lower portion of the display. Each entry has a button which can be clicked to open that tables complete description.

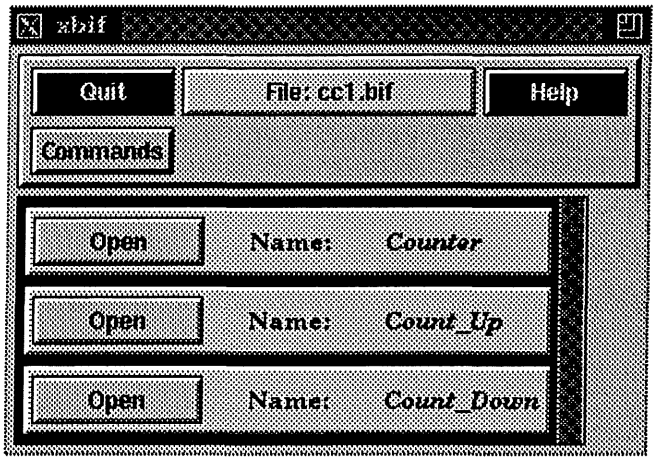

Figure 1: Top Level Window

The *Commands* button can be selected to display a menu of functions. Figure 2 shows the top level menu.

Save writes out the BIF description in textual format.

Restore reads in the original file (if it exists) discarding all changes that were made.

Add Table appends a new table entry to the existing description.

BIF-to-VHDL executes the BIF-to-VHDL ([HaCD90]) translator on the current description.

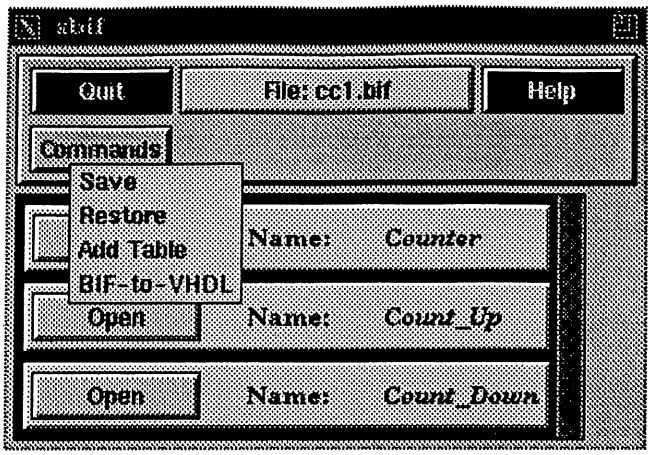

Figure 2: The Top Level Menu

1.2 The Table Window

When a button is clicked in an entry in the top level XBIF window the table corresponding to that entry is displayed in a new window. Figure 3 shows a table window.

The *Close* button closes the table window. Next to it, the table name is displayed in a user editable text box. The *Commands* button can be selected to display a menu of functions local to this table.

Check Syntax performs a syntax check on the entire table. If an error is detected it will be highlighted where it occurs. The second state of the table shown in Figure 3, *load_creg*, has two actions, $CREG = CBUS$ and $DONE = 0$. If an extra comma is added to the first line and *syntax check* is selected, Figure 4 shows how the error is highlighted.

Delete Table removes this table from the design.

Add State appends a new state template to this table.

All states in the BIF table are displayed in a scrollable viewport below the title. Each state template is composed of a name field and any number of triplet fields. The *Commands* button to the right of the name field displays a menu of functions local to the state.

Check Syntax performs a local syntax check on the entry's name field and triplets. If an error is detected it will be highlighted at the point it occurs.

Delete State removes this state from the current table.

Add Triplet appends a new triplet to this state.

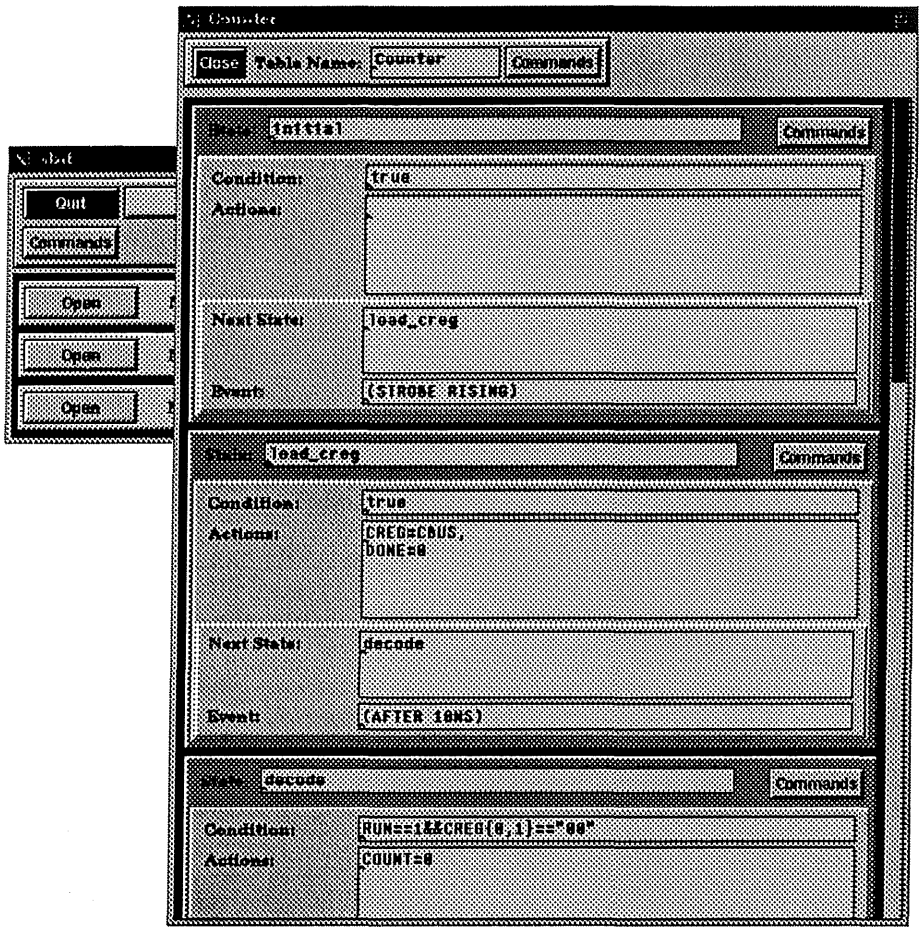

Figure 3: The Table Window

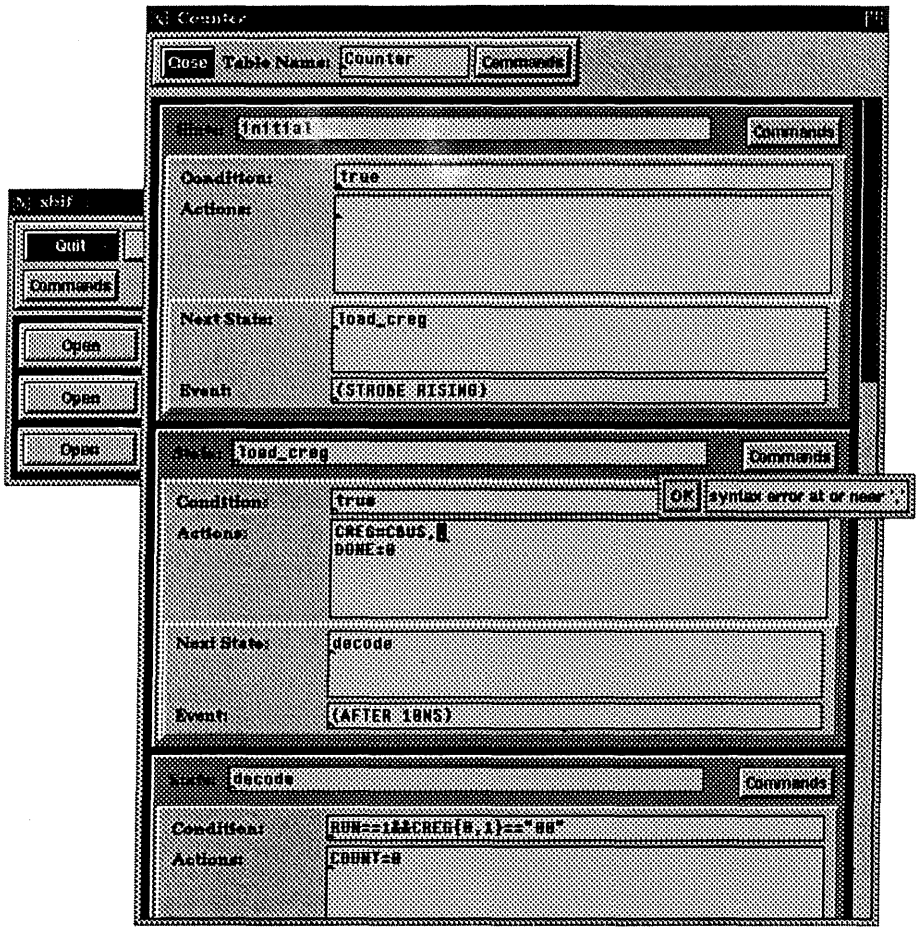

Figure 4: Table Syntax Error Example

Each triplet is made up of a condition field, an actions field and a next-state event pair. All fields are text widgets which can be edited with emacs styled commands.

2 The BIF Programming Library

2.1 Introduction

This section describes the BIF data structure and the routines that perform creation, modification, query, input and output of all objects associated with it. The routines have been designed, as much as possible, to hide the internal workings of the data structure. This has been done so that applications using these routines will not require major modifications after foreseeable updates to the BIF language and data structure.

The BIF data structure routines simplify the problem of maintaining the textual syntax. Applications need only to create the data structure and then call an appropriate routine for textual output. Since both textual input and output are handled by the BIF data structure routines, changes to the BIF language affect them only.

2.2 Compilation and Linking Details

Since we are working extensively with both sun3 and sun4 architectures, two directories will exist, each containing the BIF data structure library compiled for that architecture. The libraries are

/ch/ue/benchmarks/BIF/lib/sun3/libBIF.a
/ch/ue/benchmarks/BIF/lib/sun4/libBIF.a

For an example, an application source file using the BIF data structure routines for the sun3 architecture should be compiled :

cc -I*/ch/ue/benchmarks/BIF/include* -L*/ch/ue/benchmarks/lib/sun3* -l*BIF* [filename.c]

One file of definitions and type definitions, **BIF.h** , should be included in each application source file. The include file is located in */ch/ue/benchmarks/BIF/include.*

It is also necessary to to add the declarations:

int error_action;
int debug_value;

to your source program. These variables convey information to debugging routines, *dprintf()*, *error_msg()*, *error_msg_severe()*, (not currently documented) in the BIF library. Any value of *debug_value* higher than zero will cause routines to print out large amounts of useless information. Setting *error_action* to one will cause the program to dump core if a severe error occurs in the library.

Several examples can be found in the directory */ch/ue/benchmarks/BIF/examples.* See the file *README* for explanations. The *Makefile* in that directory shows how a sample application program is compiled.

2.3 Text File Input

The BIF library provides a parsing routine to read in a textual BIF syntax file.

File_PTR **BIF_parser** (*fi,filename*)
FILE **fi*;
char **filename*;

Before calling this routine the application program must first open the BIF syntax file (with *fopen()* from the standard C library). Then, the file descriptor must be passed to this routine, along with an optional file name. The returned value is a *File* object representing the BIF text file which can then be queried, modified, copied, output, etc. The *File* object is discussed later on below.

2.4 Text File Output

The BIF library also provides text output routines for the BIF data structure.

void **BIF_output_text** (*fo,file*)
FILE **fo*;
File_PTR *file*;

Before calling this routine the application program must first open the desired output destination file (with *fopen()* from the standard C library). Then, the file descriptor must be passed to this routine, along with a *File* object.

2.5 The Data Structure

Figures 5 and 6 show the organization of the objects in the BIF data structure. The arrows indicate that the object is part of a doubly linked list. The asterisk (*) indicates that the object is further defined later in the diagram. Dotted lines or boxes indicate optional representations. For example, the *Expression* object can have an else action list.

Each object can be further broken down into specific fields. For instance, the object *Table* has a field indicating whether or not it is a concurrent table. Specific fields of objects are described later on.

Objects are general. This means that that all fields or objects contained within a single object may not make sense in every possible instance. For example, the *AssignDelay* object may contain the object *Event*, which, in turn, contains the object *Timing*. Since *AssignDelay* already has the object *Timing*, it probably would not make sense to have Event use that object. In this particular instance, the *Expression* object of *Event* would be used. This generality is useful due to the developmental nature of the BIF language.

Each object labeled in figures 5 and 6 has associated routines that will create, query, and free it, or add it to another object. The following sections will describe each object, starting from the innermost (and therefore simplest) level of hierarchy, and proceeding to the outer levels. Each section is headed with the name of the object, a description of the object, and a list of routines that manipulate it.

All routines described are passed (and return) pointers to objects, pointers to character strings, integers, or boolean values. Boolean values are interpreted as: zero = FALSE and non-zero = TRUE, but in the interest of clarity, they will be typed Bool in these descriptions.

All function/procedure descriptions take the following form:

type returned **function or procedure name** (*parameters*)
parameter type *parameter name*;

Object types take the form *object-name_PTR*. In the include file (mentioned above) all types are cast from (int *). This is to insure that the fields of the data structure records can not be accessed by the application program.

2.6 StateIdent

The *StateIdent* object indicates a state identifier. It is described by name and an optional table that contains it. There is also a field first that indicates if this state is the first or initial state in a the table.

StateIdent_PTR **BIF_create_StateIdent** (*name,table,first*)
char **name*;
char **table*;
Bool *first*;

Returns a pointer to a *StateIdent* object having name, optional table and first value. The character strings passed are copied.

void **BIF_free_StateIdent** (*stateident*)
StateIdent_PTR *stateident*;

Frees all memory associated with a *StateIdent* object.

StateIdent_PTR **BIF_copy_StateIdent** (*stateident*)
StateIdent_PTR *stateident*;

Duplicates a *StateIdent* object and all memory associated with it.

StateIdent_PTR **BIF_modify_StateIdent** (*change,name,table,first*)
StateIdent_PTR *change*;
char **name*;
char **table*;
Bool *first*;

Modifies any record in a *StateIdent* object. Only non-null fields are considered.

char ***BIF_query_StateIdent_name** (*stateident*)
StateIdent_PTR *stateident*;

Returns a pointer to the name character string associated with the *StateIdent* object.

char ***BIF_query_StateIdent_table** (*stateident*)
StateIdent_PTR *stateident*;

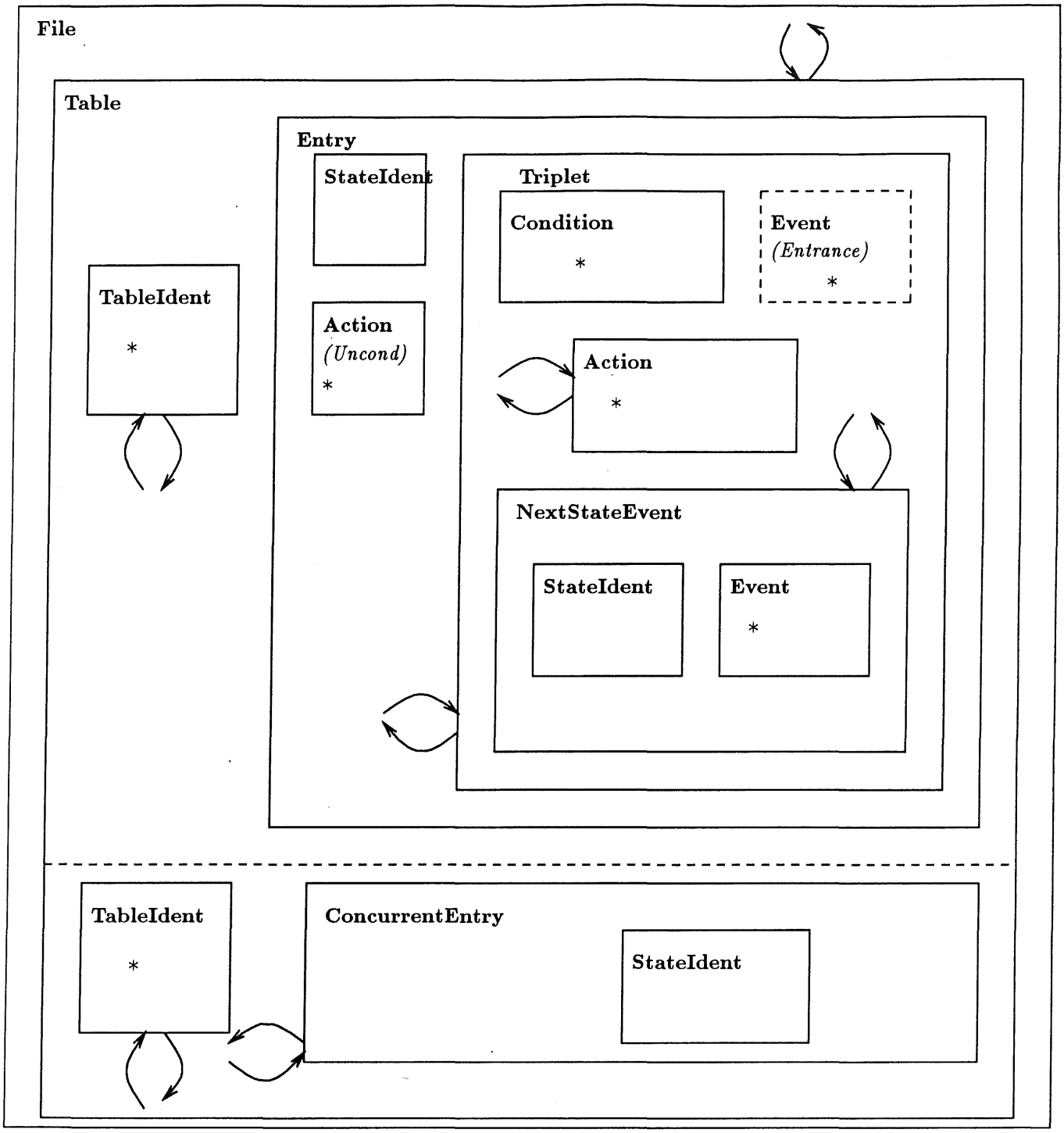

Figure 5: The BIF Data Structure (1 of 2)

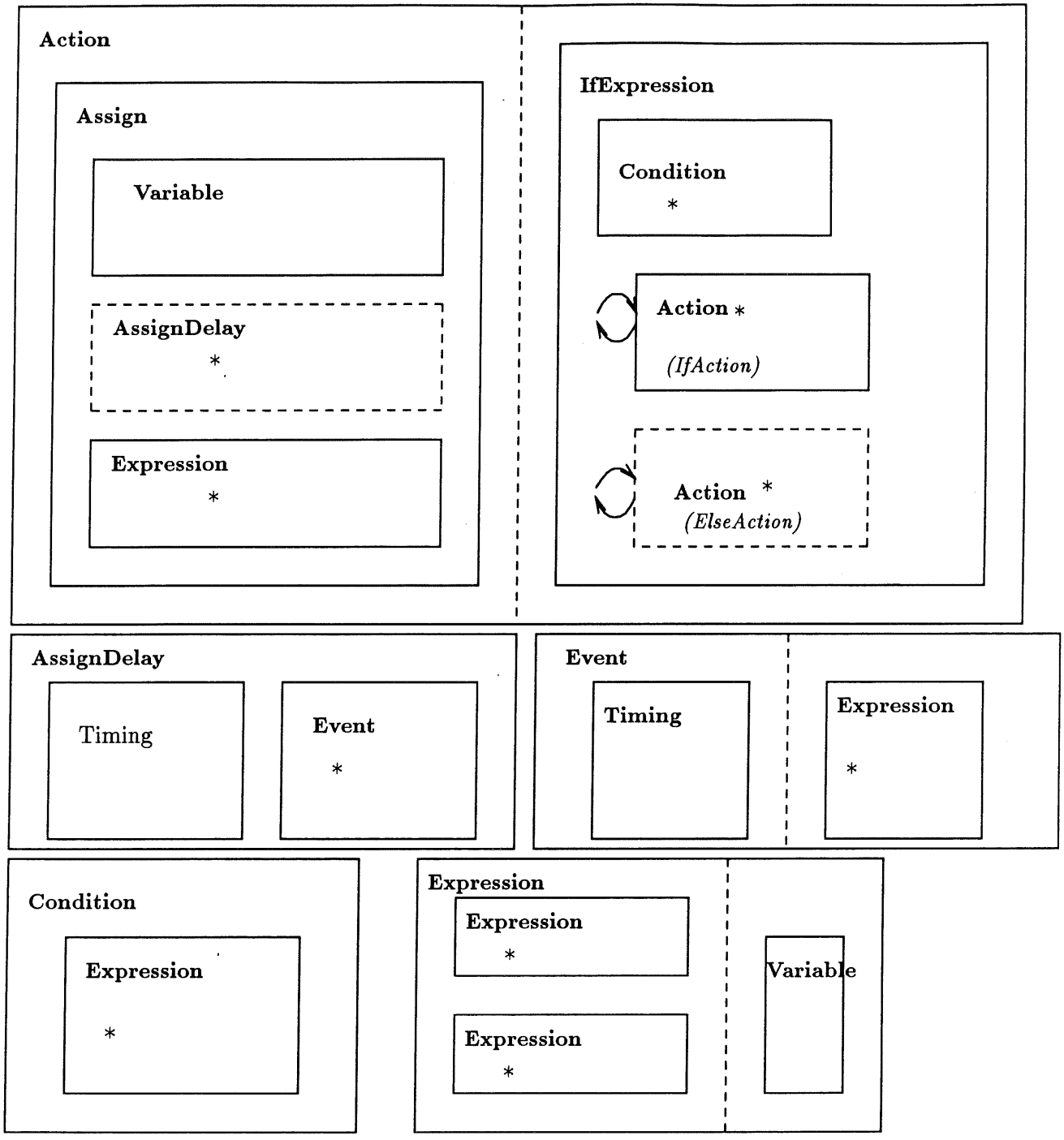

Figure 6: The BIF Data Structure (2 of 2)

Returns a pointer to the table character string associated with the *StateIdent* object.

Bool **BIF_query_StateIdent_FIRST** (*stateident*)
StateIdent_PTR *stateident*;

Returns a boolean value that indicates whether the StateIdent object is the first or initial one in the table.

2.7 TableIdent

The *TableIdent* object describes a table identifier. It is defined by name and optional *ofstate* and *oftable*. These last two provide information about a table at a higher level of hierarchy that contains this one. The *TableIdent* object is a list. This provides a complete list of table hierarchies to each table at any level of hierarchy.

TableIdent_PTR **BIF_create_TableIdent** (*name,ofstate,oftable*)
char **name*;
char **ofstate*;
char **oftable*;

Returns a pointer to a *TableIdent* object having *name*, optional *ofstate* name and *oftable* name. The character strings passed are copied.

void **BIF_free_TableIdent** (*tableident*)
TableIdent_PTR *tableident*;

Frees all memory associated with a *TableIdent* object.

TableIdent_PTR **BIF_copy_TableIdent** (*tableident*)
TableIdent_PTR *tableident*;

Duplicates a *TableIdent* object and all memory associated with it.

TableIdent_PTR **BIF_modify_TableIdent** (*change,name,ofstate,oftable,next,prev*)
TableIdent_PTR *change*;
char **name*;
char **ofstate*;
char **oftable*;
TableIdent_PTR *next*;
TableIdent_PTR *prev*;

Modifies any record in a *TableIdent* object. Only non-null fields are considered.

char ***BIF_query_TableIdent_name** (*tableident*)
TableIdent_PTR *tableident*;

Returns a pointer to the name character string associated with the *TableIdent* object.

char ***BIF_query_TableIdent_ofstate** (*tableident*)

TableIdent_PTR *tableident*;

Returns a pointer to the ofstate character string associated with the *TableIdent* object.

char ***BIF_query_TableIdent_oftable** (*tableident*)
TableIdent_PTR *tableident*;

Returns a pointer to the oftable character strings associated with the *TableIdent* object.

TableIdent_PTR **BIF_query_TableIdent_next** (*tableident*)
TableIdent_PTR *tableident*;

Returns the next TableIdent node in a possible link list of *TableIdent* objects. A return value of 0 (TableIdent_PTR 0) indicates no next node.

TableIdent_PTR **BIF_query_TableIdent_prev** (*tableident*)
TableIdent_PTR *tableident*;

Returns the previous TableIdent node in a possible link list of *TableIdent* objects. A return value of 0 (TableIdent_PTR 0) indicates no previous node.

Table_PTR **BIF_add_TableIdent_to_Table** (*tableident,table*)
TableIdent_PTR *tableident*;
Table_PTR *table*;

Adds a *TableIdent* object to a table. A pointer to the same table is returned. Since the *TableIdent* object is a linked list, successive adds append the *TableIdent* object to the end of the list. See *Table* object description for more details.

2.8 Condition

The *Condition* object describes a conditional expression. It is defined by an *Expression* object or by a boolean value indicating that it is an else condition. In the latter case, it is assumed that other conditions precede it.

Cond_PTR **BIF_create_Condition** (*else,expr*)
Bool *else*;
Expr_PTR *expr*;

Returns a pointer to a *Condition* object having the expression expr, or the else meaning, indicated with the parameter else.

void **BIF_free_Condition** (*condition*)
Cond_PTR *condition*;

Frees all memory associated with a *Condition* object.

Cond_PTR **BIF_copy_Condition** (*condition*)
Cond_PTR *condition*;

Duplicates a *Condition* object and all memory associated with it.

Condition_PTR **BIF_modify_Condition** (*change,velse,expr*)
Condition_PTR *change*;
Bool *else*;
Expr_PTR *expr*;

Modifies any record in a *Condition* object. Only non-null fields are considered.

Bool **BIF_query_Condition_ELSE** (*condition*)
Cond_PTR *condition*;

Returns a boolean value indicating whether or not the Condition object is an else condition.

Expr_PTR **BIF_query_Condition_expression** (*condition*)
Cond_PTR *condition*;

Returns an *Expression* object that corresponds to the *Condition* object. If the *Condition* object happens to be an else condition the value returned is 0 (Cond_PTR 0).

2.9 IfExpression

The If*Expression* object denotes an if-then-else construct. It is used as part of the *Action* object. It is defined with a *Condition* object, *if_cond*, and two *Action* linked list objects, *if_actions* and *else_actions*. The *else_actions* list can be empty.

IfExpr_PTR **BIF_create_IfExpression** (*if_cond,if_actions,else_actions*)
Cond_PTR *if_cond*;
Action_PTR *if_actions*;
Action_PTR *else_actions*;

Returns a pointer to a *IfExpression* object having *if_cond*, as the if–then condition, *if_actions*, and *else_actions*. The latter can be 0 (Action_PTR 0).

void **BIF_free_IfExpression** (*if_expr*)
IfExpr_PTR *if_expr*;

Frees all memory associated with a If*Expression* object.

IfExpr_PTR **BIF_copy_IfExpression** (*ifexpr*)
IfExpr_PTR *ifexpr*;

Duplicates a *IfExpression* object and all memory associated with it.

IfExpr_PTR **BIF_modify_IfExpression** (*change,if_cond,if_actions,else_actions*)
IfExpr_PTR *change*;
Cond_PTR *if_cond*;
Action_PTR *if_actions*;
Action_PTR *else_actions*;

Modifies any record in a *IfExpression* object. Only non-null fields are considered.

Cond_PTR **BIF_query_IfExpression_if_cond** (*if_expr*)
IfExpr_PTR *if_expr*;

Returns a pointer to a *Condition* object corresponding to the if-then condition of the *IfExpression* object.

int **BIF_query_IfExpression_num_if_actions** (*if_expr*)
IfExpr_PTR *if_expr*;

Returns the number of *Action* objects contained in the if–then actions list of the *IfExpression* object.

Action_PTR **BIF_query_IfExpression_if_actions** (*if_expr*)
IfExpr_PTR *if_expr*;

Returns an *Action* object (may be the head of a list) corresponding to the if–then actions of the *IfExpression* object.

int **BIF_query_IfExpression_num_else_actions** (*if_expr*)
IfExpr_PTR *if_expr*;

Returns the number of *Action* objects contained in the else actions list of the *IfExpression* object.

Action_PTR **BIF_query_IfExpression_else_actions** (*if_expr*)
IfExpr_PTR *if_expr*;

Returns an *Action* object (may be the head of a list) corresponding to the else actions of the *IfExpression* object.

2.10 Event

The *Event* object describes either delay durations, signal event expressions, or special-case hierarchical calls.

Event_PTR **BIF_create_Event** (*expression*)
Expr_PTR *expression*;

Returns an *Event* object that describes an event expression.

Event_PTR **BIF_create_CALL_Event**()

Returns an *Event* object that denotes a call to a lower level of hierarchy.

Event_PTR **BIF_create_DELAY_Event** (*delay*)
Timing_PTR *delay*;

Returns an *Event* object that denotes a delay duration constraint on a triplet. This is not to be confused with **BIF_create_AFTER_DELAY_Event** which indicates the amount of time to delay before proceeding to the next state.

Event_PTR **BIF_create_AFTER_DELAY_Event** (*delay*)
Timing_PTR *delay*;

Returns an *Event* object that indicates the amount of time to delay before proceeding to the next state. This is not to be confused with **BIF_create_DELAY_Event** which denotes a delay duration constraint on a triplet.

void **BIF_free_Event** (*event*)
Event_PTR *event*;

Frees all memory associated with a *Event* object.

Event_PTR **BIF_copy_Event** (*event*)
Event_PTR *event*;

Duplicates an *Event* object and all memory associated with it.

Event_PTR **BIF_modify_Event** (*change,expr*)
Event_PTR *change*;
Expr_PTR *expr*;

Modifies any record in an *Event* object. Only non-null fields are considered.

Event_PTR **BIF_modify_DELAY_Event** (*change,delay*)
Event_PTR *change*;
Timing_PTR *delay*;

Modifies the delay in a DELAY *Event* object. Only non-null fields are considered.

Event_PTR **BIF_modify_AFTER_DELAY_Event** (*change,delay*)
Event_PTR *change*;
Timing_PTR *delay*;

Modifies the delay in an AFTER DELAY *Event* object. Only non-null fields are considered. See above for further discussion of DELAY and AFTER DELAY events.

Bool **BIF_query_Event_CALL** (*event*)
Event_PTR *event*;

Returns a boolean value that indicates whether or not this *Event* object denotes a call to a lower level of hierarchy.

Bool **BIF_query_Event_DELAY** (*event*)
Event_PTR *event*;

Returns a boolean value that indicates whether or not this *Event* object denotes a delay duration constraint on a triplet.

Bool **BIF_query_Event_AFTER_DELAY** (*event*)
Event_PTR *event*;

Returns a boolean value that indicates whether or not this *Event* object indicates the amount of time to delay before proceeding to the next state.

Bool **BIF_query_Event_EXPRESSION** (*event*)
Event_PTR *event*;

Returns a boolean value that indicates whether or not this *Event* object denotes an expression of signal events.

Timing_PTR **BIF_query_Event_delay** (*event*)
Event_PTR *event*;

Returns a *Timing* object for both types, DELAY and AFTER_DELAY, of the *Event* object.

Expr_PTR **BIF_query_Event_expression** (*event*)
Event_PTR *event*;

Returns an *Expression* object that contains an expression of signal events for the *Event* object.

2.11 Timing

The *Timing* object describes delay duration, the units the delay value is expressed in, nano-seconds or micro-seconds, and a constraint on the delay, maximum, minimum, or nominal.

Timing_PTR **BIF_create_Timing** (*nom,min,max,delay,nano_secs,micro_secs*)
Bool *num*;
Bool *min*;
Bool *max*;
int *delay*;
Bool *nano_secs*;
Bool *micro_secs*;

Returns a pointer to a *Timing* object having the values specified. Note that having more than one of *nom*, *min*, *max*, TRUE at one time does not make sense.

void **BIF_free_Timing** (*timing*)
Timing_PTR *timing*;

Frees all memory associated with a *Timing* object.

Timing_PTR **BIF_copy_Timing** (*timing*)
Timing_PTR *timing*;

Duplicates a *Timing* object and all memory associated with it.

Timing_PTR **BIF_modify_Timing** (*change,nom,min,max,delay,ns,ms*)
Timing_PTR *change*;
Bool *nom*;

Bool *min*;
Bool *max*;
int *delay*;
Bool *ns*;
Bool *ms*;

Modifies any record in a *Timing* object. Boolean fields that are false are ignored. A *delay* value of -1 causes the original value to remain unchanged.

Bool **BIF_query_Timing_MAX** (*timing*)
Timing_PTR *timing*;

Returns whether or not the delay constraint is specified maximum.

Bool **BIF_query_Timing_MIN** (*timing*)
Timing_PTR *timing*;

Returns whether or not the delay constraint is specified minimum.

Bool **BIF_query_Timing_NOM** (*timing*)
Timing_PTR *timing*;

Returns whether or not the delay constraint is specified nominal.

int **BIF_query_Timing_delay** (*timing*)
Timing_PTR *timing*;

Returns the value of the delay for the *Timing* object.

Bool **BIF_query_Timing_NS** (*timing*)
Timing_PTR *timing*;

Returns whether or not the delay is in nano-seconds.

Bool **BIF_query_Timing_MS** (*timing*)
Timing_PTR *timing*;

Returns whether or not the delay is in micro-seconds.

2.12 AssignDelay

The *AssignDelay* object denotes a delayed assignment to a variable optionally after some event. It is used in the *Assign* object.

AssignDelay_PTR **BIF_create_AssignDelay** (*delay,event*)
Timing_PTR *delay*;
Event_PTR *event*;

Returns a pointer to an *AssignDelay* object having the delay specified by a *Timing* object, and the optional event specified in an *Event* object.

void **BIF_free_AssignDelay** (*assigndelay*)
AssignDelay_PTR *assigndelay*;

Frees all memory associated with an *AssignDelay* object.

AssignDelay_PTR **BIF_copy_AssignDelay** (*assigndelay*)
AssignDelay_PTR *assigndelay*;

Duplicates an *AssignDelay* object and all memory associated with it.

AssignDelay_PTR **BIF_modify_AssignDelay** (*change,delay,event*)
AssignDelay_PTR *change*;
Timing_PTR *delay*;
Event_PTR *event*;

Modifies any record in an *AssignDelay* object. Only non-null fields are considered.

Timing_PTR **BIF_query_AssignDelay_delay** (*assigndelay*)
AssignDelay_PTR *assigndelay*;

Returns a *Timing* object which corresponds to the delay specified in the *AssignDelay* object.

Event_PTR **BIF_query_AssignDelay_event** (*assigndelay*)
AssignDelay_PTR *assigndelay*;

Returns an *Event* object which corresponds to the event after which the delay is to occur. If there is no event in the *AssignDelay* object then a NULL ((AssignDelay_PTR) 0) value is returned.

2.13 Variable

The *Variable* object represents a variable. It can be a selection, in which case, values for the start and end of the selection are specified, an array reference, or a plain variable or constant. See the section on the *Expression* object for further routines that create and return variables.

Variable_PTR **BIF_create_EVENT_Variable** (*name,array_ref,rising,falling*)
char **name*;
int *array_ref*;
Bool *rising*;
Bool *falling*;

Returns a *Variable* object describing an event having character string *name*, and specification *rising* or *falling*. *array_ref* makes it possible to reference arrays of signal events. IMPORTANT: For single events this value should be -1. Note also that only one of rising and falling should be TRUE at a time.

Variable_PTR **BIF_create_CONSTANT_Variable** (*value*)
int *value*;

Returns a *Variable* object having the value of a constant. Currently, only 32bit integer decimal values are supported.

Variable_PTR **BIF_create_Variable** (*name,start,stop,array_ref*)
char **name*;
int *start*;
int *stop*;
int *array_ref*;

Returns a *Variable* object describing a non-event variable having character string *name*. *start* and *stop* allow specification of selection within a vectored variable. *start* indicates the numerical starting value and *stop* the ending value. For variables that do not have selection specification, both of these values should be -1. *array_ref* specifies the value of an array reference. For variables that do not require array referencing, this value should be -1.

void **BIF_free_Variable** (*variable*)
Variable_PTR *variable*;

Frees all memory associated with a *Variable* object.

Variable_PTR **BIF_copy_Variable** (*variable*)
Variable_PTR *variable*;

Duplicates a *Variable* object and all memory associated with it.

Variable_PTR **BIF_modify_Variable** (*change,name,start,stop,arrayref*)
Variable_PTR *change*;
char **name*;
int *start*;
int *stop*;
int *arrayref*;

Modifies any record in a *Variable* object. Integer values that are not to be changed should have the value -1.

Variable_PTR **BIF_modify_EVENT_Variable** (*change,name,arrayref,rising,falling*)
Variable_PTR *change*;
char **name*;
int *arrayref*;
Bool *rising*;
Bool *falling*;

Modifies any record in a *Variable* object. Integer values that are not to be changed should have the value -1. Bool values that are to remain unchanged should be false.

Variable_PTR **BIF_modify_CONSTANT_Variable** (*change,value*)
Variable_PTR *change*;

int *value*;

Modifies a CONSTANT *Variable* object. Despite the contradiction in terms, a CONSTANT variable has a constant integer value.

char ***BIF_query_Variable_name** (*variable*)
Variable_PTR *variable*;

Returns a character string corresponding to the name of *Variable* object.

Bool **BIF_query_Variable_EVENT** (*variable*)
Variable_PTR *variable*;

Returns a boolean value indicating whether or not the variable is an event.

Bool **BIF_query_Variable_SELECTION** (*variable*)
Variable_PTR *variable*;

Returns a boolean value indicating whether or not the variable has selection values, i.e, start and stop values.

Bool **BIF_query_Variable_CONSTANT** (*variable*)
Variable_PTR *variable*;

Returns a boolean value indicating whether or not the variable is a constant.

Bool **BIF_query_Variable_ARRAYREF** (*variable*)
Variable_PTR *variable*;

Returns a boolean value indicating whether or not the variable has an array reference value.

int **BIF_query_Variable_select_start** (*variable*)
Variable_PTR *variable*;

Returns the starting selection value for the *Variable* object. If the *Variable* object is not of a selection type the value -1 is returned.

int **BIF_query_Variable_select_stop** (*variable*)
Variable_PTR *variable*;

Returns the ending selection value for the *Variable* object. If the *Variable* object is not of a selection type the value -1 is returned.

int **BIF_query_Variable_array_ref** (*variable*)
Variable_PTR *variable*;

Returns the array reference value for the *Variable* object. if the *Variable* object does not have an array reference value the value -1 is returned.

Bool **BIF_query_Variable_EVENT_RISING** (*variable*)
Variable_PTR *variable*;

Returns a boolean value indicating whether or not the event corresponding to the *Variable* object is rising. If the *Variable* object is not of type "event", False is returned automatically.

Bool **BIF_query_Variable_EVENT_FALLING** (*variable*)
Variable_PTR *variable*;

Returns a boolean value indicating whether or not the event corresponding to the *Variable* object is falling. If the Variable object is not of type event False is returned automatically.

2.14 Assign

The *Assign* object denotes an assignment. The assignment has a left-hand side *Variable* object, an optional *AssignDelay* object, and a right-hand side *Expression* object.

Assign_PTR **BIF_create_Assign** (*lhs,assigndelay,rhs*)
Variable_PTR *lhs*;
AssignDelay_PTR *assigndelay*;
Expr_PTR *rhs*;

Returns a pointer to a *Assign* object having *lhs* as the left-hand side, *assigndelay* as an option assignment delay, and *rhs* as the right-hand side.

void **BIF_free_Assign** (*assign*)
Assign_PTR *assign*;

Frees all memory associated with an *Assign* object.

Assign_PTR **BIF_copy_Assign** (*Assign*)
Assign_PTR *assign*;

Duplicates a *Assign* object and all memory associated with it.

Assign_PTR **BIF_modify_Assign** (*change,lhs,delay,rhs*)
Assign_PTR *change*;
Variable_PTR *lhs*;
AssignDelay_PTR *delay*;
Expr_PTR *rhs*;

Modifies any record in an *Assign* object. Only non-null fields are considered.

Variable_PTR **BIF_query_Assign_lhs** (*assign*)
Assign_PTR *assign*;

Returns a *Variable* object corresponding to the left-hand side of the assignment in the *Assign* object.

AssignDelay_PTR **BIF_query_Assign_assign_delay** (*assign*)
Assign_PTR *assign*;

Returns an *AssignDelay* object, if it exists, for the *Assign* object. 0 (AssignDelay_PTR 0), otherwise.

Expr_PTR **BIF_query_Assign_rhs** (*assign*)
Assign_PTR *assign*;

Returns an *Expression* object corresponding to the right-hand side of the *Assign* object.

2.15 Action

The *Action* object denotes an action in a table and/or triplet. It can be either an assignment, in which case it has an *Assign* object, or an if-then-else expression, in which case it contains an *IfExpression* object. Note that *IfExpression* objects also have actions. This provides arbitrary levels of nested if-then-else-statements. The *Action* object is a list. This provides multiple assignments to be described in a single action entry.

Action_PTR **BIF_create_IF_Action** (*ifexpr*)
IfExpr_PTR *ifexpr*;

Returns an *Action* object containing the *IfExpression* object passed to it.

Action_PTR **BIF_create_ASSIGN_Action** (*assign*)
Assign_PTR *assign*;

Returns an *Action* object containing the *Assign* object passed to it.

void **BIF_free_Action** (*action*)
Action_PTR *action*;

Frees all memory associated with an *Action* object.

Action_PTR **BIF_copy_Action** (*action*)
Action_PTR *action*;

Duplicates a *Action* object and all memory associated with it.

Action_PTR **BIF_modify_Action** (*change,assign,next,prev*)
Action_PTR *change*;
Assign_PTR *assign*;
Action_PTR *next*;
Action_PTR *prev*;

Modifies a record in an *Action* object.

Action_PTR **BIF_modify_IF_Action** (*change,ifexpr,next,prev*)
Action_PTR *change*;
IfExpr_PTR *ifexpr*;
Action_PTR *next*;

Action_PTR *prev*;

Modifies a record in an *Action* object if that object contains an if-statement in the form of an *IfExpression* object.

Bool **BIF_query_Action_IF** (*action*)
Action_PTR *action*;

Returns whether or not this *Action* object contains an *IfExpression* object.

Bool **BIF_query_Action_ASSIGN** (*action*)
Action_PTR *action*;

Returns whether or not this *Action* object contains an *Assign* object. Note that combining this routine with **BIF_query_Action_IF** is redundant since an action can only be one of two things. However, later additions to BIF may allow *Action* objects to contain unit declarations, making this routine required.

IfExpr_PTR **BIF_query_Action_if_expression** (*action*)
Action_PTR *action*;

Returns an *IfExpression* object for the given *Action* object. 0 (IfExpr_PTR 0) is returned if it does not exist.

Assign_PTR **BIF_query_Action_assign** (*action*)
Action_PTR *action*;

Returns an *Assign* object for the given *Action* object. 0 (Assign_PTR 0) is returned if it does not exist.

Action_PTR **BIF_query_Action_next** (*action*)
Action_PTR *action*;

Returns the next *Action* object in a possible linked list of *Action* objects. A return value of 0 (Action_PTR 0) indicates no next object.

Action_PTR **BIF_query_Action_prev** (*action*)
Action_PTR *action*;

Returns the previous *Action* object in possible linked list of *Action* objects. A return value of 0 (Action_PTR 0) indicates no previous object.

IfExpr_PTR **BIF_add_IF_Action_to_IfExpression** (*action,ifexpr*)
Action_PTR *action*;
IfExpr_PTR *ifexpr*;

Adds an *Action* object to the if-then actions in a *IfExpression* object. A pointer to the same *IfExpression* object is returned. Since the *Action* object is a linked list, successive adds append the *Action* object to the end of the list. See *IfExpression* object description for more details.

IfExpr_PTR **BIF_add_ELSE_Action_to_IfExpression** (*action,ifexpr*)

Action_PTR *action*;
IfExpr_PTR *ifexpr*;

Adds an *Action* object to the else actions in a *IfExpression* object. A pointer to the same *IfExpression* object is returned. Since the *Action* object may be part of a linked list, successive adds append the *Action* object to the end of the list. See *IfExpression* object description for more details.

Triplet_PTR **BIF_add_Action_to_Triplet** (*action,triplet*)
Action_PTR *action*;
Triplet_PTR *triplet*;

Adds an *Action* object to the action field of a *Triplet* object. A pointer to the same *Triplet* object is returned. Since the *Action* object may be part of a linked list, successive adds append the *Action* object to the end of the list. See *Triplet* object description for more details.

Entry_PTR **BIF_add_UNCOND_Action_to_Entry** (*action,entry*)
Action_PTR *action*;
Entry_PTR *entry*;

Adds an *Action* object to the unconditional actions field of an *Entry* object. A pointer to the same *Entry* object is returned. Since the *Action* object may be part of a linked list, successive adds append the *Action* object to the end of the list. See *Entry* object description for more details.

2.16 NextStateEvent

The *NextStateEvent* object describes the next state to proceed to in a given entry, and the event that causes the transition. The next state is described by a *StateIdent* object, and the event is described by an *Event* object. In a special case of triplets, the triplet is be executed serially (sequentially) with respect to its neighboring triplets. In this case the *NextStateEvent* object has a serial indicator, the next state value is undefined, and the event may be a duration constraint on the triplet. The *NextStateEvent* object is a list. This allows multiple next state events to be specified in a single triplet.

NextStateEvent_PTR **BIF_create_SERIAL_NextStateEvent** (*event*)
Event_PTR *event*;

Returns a *NextStateEvent* object having the serial meaning described above. Note that, in this case, the *Event* object should specify a delay constraint. At this time, no checking is done to enforce this.

NextStateEvent_PTR **BIF_create_NextStateEvent** (*next_state,event*)
StateIdent_PTR *next_state*;
Event_PTR *event*;

Returns a *NextStateEvent* object having the given *StateIdent* object next-state, and the *Event* object passed. Note that, for consistency, this routine might have been called *BIF_create_PARALLEL_NextState* but since parallel execution is the default, therefore unspecified, this name was not used.

void **BIF_free_NextStateEvent** (*next*)

NextStateEvent_PTR *next*;

Frees all memory associated with the *NextStateEvent* object.

NextStateEvent_PTR **BIF_copy_NextStateEvent** (*next*)
NextStateEvent_PTR *next*;

Duplicates a *NextStateEvent* object and all memory associated with it.

NextStateEvent_PTR **BIF_modify_NextStateEvent** (*change,nextstate,event,next,prev*)
NextStateEvent_PTR *change*;
StateIdent_PTR *nextstate*;
Event_PTR *event*;
NextStateEvent_PTR *next*;
NextStateEvent_PTR *prev*;

Modifies any record in a *NextStateEvent* object. Only non-null fields are considered.

NextStateEvent_PTR **BIF_modify_SERIAL_NextStateEvent** (*change,nextstate,next,prev*)
NextStateEvent_PTR *change*;
StateIdent_PTR *nextstate*;
NextStateEvent_PTR *next*;
NextStateEvent_PTR *prev*;

Modifies any record in a SERIAL *NextStateEvent* object. Only non-null fields are considered. See above for further discussion of SERIAL.

Bool **BIF_query_NextStateEvent_SERIAL** (*next*)
NextStateEvent_PTR *next*;

Returns a boolean value indicating whether or not this *NextStateEvent* object has the serial meaning described above. Logically, there should also exist a routine *BIF_query_NextStateEvent_PARALLEL*, but does not, for reasons discussed above (sheer laziness).

StateIdent_PTR **BIF_query_NextStateEvent_next_state** (*next*)
NextStateEvent_PTR *next*;

Returns a *StateIdent* object containing the next state specification for the given *NextStateEvent* object.

Event_PTR **BIF_query_NextStateEvent_event** (*next*)
NextStateEvent_PTR *next*;

Returns an *Event* object containing the event specification for the given *NextStateEvent* object.

NextStateEvent_PTR **BIF_query_NextStateEvent_next** (*next*)
NextStateEvent_PTR *next*;

Returns the next *NextStateEvent* object in a possible linked list of *NextStateEvent* objects. A return value of 0 (NextStateEvent_PTR 0) indicates no next node.

NextStateEvent_PTR **BIF_query_NextStateEvent_prev** (*next*)
NextStateEvent_PTR *next*;

Returns the previous *NextStateEvent* object in a possible linked list of *NextStateEvent* objects. A return value of 0 (NextStateEvent_PTR 0) indicates no next node.

Triplet_PTR **BIF_add_NextStateEvent_to_Triplet** (*next_state,triplet*)
NextStateEvent_PTR *next_state*;
Triplet_PTR *triplet*;

Adds a *NextStateEvent* object to a triplet. A pointer to the same triplet is returned. Since the *NextStateEvent* object is a linked list, successive adds append the *NextStateEvent* object to the end of the list. See *Triplet* object description for more details.

2.17 Triplet

The *Triplet* object contains a *Condition* object, *Action* object, and *NextStateEvent* object. Recall that both *Action* and *NextStateEvent* objects are linked lists. This means that triplets contain multiple actions and next-state-event pairs. In addition, to provide some compatibility with different state machine modeling, the *Triplet* object has an *Event* object which allows specification of entrance events. The *Triplet* object is also a list.

Triplet_PTR **BIF_create_Triplet** (*entrance_events,condition,actions,next_state_events*)
Event_PTR *entrance_events*;
Cond_PTR *condition*;
Action_PTR *actions*;
NextStateEvent_PTR *next_state_events*;

Returns a *Triplet* object containing *Condition* object, an *Action* object list, and a *NextStateEvent* object list. As was mentioned above, there is can also be an *Event* object specified to contain entrance events. This field can be NULL.

void **BIF_free_Triplet** (*triplet*)
Triplet_PTR *triplet*;

Frees all memory associated with a *Triplet* object.

Triplet_PTR **BIF_copy_Triplet** (*triplet*)
Triplet_PTR *triplet*;

Duplicates a *Triplet* object and all memory associated with it.

Triplet_PTR **BIF_modify_Triplet** (*change,entrance_events,condition,actions,next_states,next,prev*
Triplet_PTR *change*;
Event_PTR *entrance_events*;
Cond_PTR *condition*;
Action_PTR *actions*;
NextStateEvent_PTR *next_states*;

Triplet_PTR *next*;
Triplet_PTR *prev*;

Modifies any record in a *Triplet* object. Only non-null fields are considered.

Event_PTR **BIF_query_Triplet_entrance_events** (*triplet*)
Triplet_PTR *triplet*;

Returns an *Event* object referring to the entrance events of the *Triplet* object. If there are no entrance events (usually, BIF uses exit events in the form of the *NextStateEvent* object), the return value is 0 (Event_PTR 0).

Cond_PTR **BIF_query_Triplet_condition** (*triplet*)
Triplet_PTR *triplet*;

Returns a *Condition* object referring to the condition on which the actions in this triplet will be performed.

int **BIF_query_Triplet_num_actions** (*triplet*)
Triplet_PTR *triplet*;

Returns the number of *Action* objects in the triplet.

Action_PTR **BIF_query_Triplet_actions** (*triplet*)
Triplet_PTR *triplet*;

Returns an *Action* object denoting the actions in this triplet. Remember that the *Action* object is a linked list, so that this list can be traversed to obtain all of the actions.

int **BIF_query_Triplet_num_next_state_events** (*triplet*)
Triplet_PTR *triplet*;

Returns the number of *NextStateEvent* objects in this triplet.

NextStateEvent_PTR **BIF_query_Triplet_next_states** (*triplet*)
Triplet_PTR *triplet*;

Returns a *NextStateEvent* object denoting the next-state-event pairs in this triplet. Remember that the *NextStateEvent* object is a linked list, so that this list can be traversed to obtain all of the actions.

Triplet_PTR **BIF_query_Triplet_next** (*triplet*)
Triplet_PTR *triplet*;

Returns the next *Triplet* object in a possible link list of Triplet objects. A return value of 0 (Triplet_PTR 0) indicates no next object.

Triplet_PTR **BIF_query_Triplet_prev** (*triplet*)
Triplet_PTR *triplet*;

Returns the previous *Triplet* object in a possible link list of *Triplet* objects. A return value of 0 (Triplet_PTR 0) indicates no previous object.

Entry_PTR **BIF_add_Triplet_to_Entry** (*triplet,entry*)
Triplet_PTR *triplet*;
Entry_PTR *entry*;

Adds a *Triplet* object to an entry. A pointer to the same *Entry* is returned. Since the *Triplet* object may be a linked list, successive adds append the *Triplet* object to the end of the list. See *Entry object* description for more details.

2.18 Entry

The *Entry* object indicates an entry in a BIF state table. It contains a *StateIdent* object indicating the present state, an optional *Action* object giving the unconditional actions for this entry, and a *Triplet* object giving the triplets for this particular entry. The *Entry* object is a list.

Entry_PTR **BIF_create_Entry** (*stateident,uncond_actions,triplets*)
StateIdent_PTR *stateident*;
Action_PTR *uncond_actions*;
Triplet_PTR *triplets*;

Returns an *Entry* object having present state *stateident*, optional unconditional actions *uncond_actions*, and triplets - *Triplet* objects.

void **BIF_free_Entry** (*entry*)
Entry_PTR *entry*;

Frees all memory associated with an Entry object.

Entry_PTR **BIF_copy_Entry** (*entry*)
Entry_PTR *entry*;

Duplicates a *Entry* object and all memory associated with it.

Entry_PTR **BIF_modify_Entry** (*change,ident,uncond_actions,triplets,next,prev*)
Entry_PTR *change*;
StateIdent_PTR *ident*;
Action_PTR *uncond_actions*;
Triplet_PTR *triplets*;
Entry_PTR *next*;
Entry_PTR *prev*;

Modifies any record in an *Entry* object. Only non-null fields are considered.

StateIdent_PTR **BIF_query_Entry_state** (*entry*)
Entry_PTR *entry*;

Returns a *StateIdent* object representing the present state of the *Entry* object.

int **BIF_query_Entry_num_uncond_actions** (*entry*)

Entry_PTR *entry*;

Returns the number of unconditional actions in the *Entry* object.

Action_PTR **BIF_query_Entry_uncond_actions** (*entry*)
Entry_PTR *entry*;

Returns an *Action* object having the unconditional actions for this *Entry* object.

int **BIF_query_Entry_num_triplets** (*entry*)
Entry_PTR *entry*;

Returns the number of triplets in the *Entry* object.

Triplet_PTR **BIF_query_Entry_triplets** (*entry*)
Entry_PTR *entry*;

Returns a *Triplet* object containing the triplets for this *Entry* object.

Entry_PTR **BIF_query_Entry_next** (*entry*)
Entry_PTR *entry*;

Returns the next *Entry* object in a possible linked list of *Entry* objects. A return value of 0 (Entry_PTR 0) indicates no next object.

Entry_PTR **BIF_query_Entry_prev** (*entry*)
Entry_PTR *entry*;

Returns the previous *Entry* object in a possible linked list of *Entry* objects. A return value of 0 (Entry_PTR 0) indicates no previous object.

Table_PTR **BIF_add_Entry_to_Table** (*entry,table*)
Entry_PTR *entry*;
Table_PTR *table*;

Adds an *Entry* object to a table. A pointer to the same table is returned. Since the *Entry* object is a linked list, successive adds append the *Entry* object to the end of the list. See *Table* object description for more details.

2.19 ConcurrentEntry

The *ConcurrentEntry* object represents concurrent entries in a concurrent table. The assumption is that all states or tables named in a concurrent table run in parallel. The *ConcurrentEntry* object is defined only by a *StateIdent* object. It is thought that more objects will be added to the concurrent table entry in the future. Like the Entry object, the *ConcurrentEntry* object is a linked list.

ConcurrentEntry_PTR **BIF_create_ConcurrentEntry** (*stateident*)
StateIdent_PTR *stateident*;

Returns a *ConcurrentEntry* object having the state and/or table name defined in the *StateIdent* object.

void **BIF_free_ConcurrentEntry** (*entry*)
ConcurrentEntry_PTR *entry*;

Frees all memory associated with the *ConcurrentEntry* object.

ConcurrentEntry_PTR **BIF_copy_ConcurrentEntry** (*entry*)
ConcurrentEntry_PTR *entry*;

Duplicates a *ConcurrentEntry* object and all memory associated with it.

ConcurrentEntry_PTR **BIF_modify_ConcurrentEntry** (*change,ident,next,prev*)
ConcurrentEntry_PTR *change*;
StateIdent_PTR *ident*;
ConcurrentEntry_PTR *next*;
ConcurrentEntry_PTR *prev*;

Modifies any record in a *ConcurrentEntry* object. Only non-null fields are considered.

StateIdent_PTR **BIF_query_ConcurrentEntry_ident** (*entry*)
ConcurrentEntry_PTR *entry*;

Returns a *StateIdent* object having the state and/or table name of this entry.

ConcurrentEntry_PTR **BIF_query_ConcurrentEntry_next** (*entry*)
ConcurrentEntry_PTR *entry*;

Returns the next *ConcurrentEntry* object in a possible linked list of *ConcurrentEntry* objects. A return value of 0 (ConcurrentEntry_PTR 0) indicates no next object.

ConcurrentEntry_PTR **BIF_query_ConcurrentEntry_prev** (*entry*)
ConcurrentEntry_PTR *entry*;

Returns the previous *ConcurrentEntry* object in a possible linked list of *ConcurrentEntry* objects. A return value of 0 (ConcurrentEntry_PTR 0) indicates no previous object.

Table_PTR **BIF_add_ConcurrentEntry_to_Table** (*entry,table*)
ConcurrentEntry_PTR *entry*;
Table_PTR *table*;

Adds a *ConcurrentEntry* object to a table. A pointer to the same table is returned. Since the *ConcurrentEntry* object is a linked list, successive adds append the *ConcurrentEntry* object to the end of the list. It is an error to add a *ConcurrentEntry* object to a non-concurrent table. See *Table* object description for more details.

2.20 Table

The *Table* object represents a state table. It is defined by a list of table identifiers, in the form of the *TableIdent* object, and entries, in the form of either Entry objects or *ConcurrentEntry* objects. The *Table* is a linked list, allowing multiple tables in a single BIF description.

Table_PTR **BIF_create_Table** (*tableident,entries*)
TableIdent_PTR *tableident*;
Entry_PTR *entries*;

Returns a *Table* object having identity *tableident* and entries represented by *entries.*

Table_PTR **BIF_create_CONCURRENT_Table** (*tableident,concurrent_entries*)
TableIdent_PTR *tableident*;
ConcurrentEntry_PTR *concurrent_entries*;

Returns a *Table* object having identity *tableident* and concurrent entries represented by *entries.*

void **BIF_free_Table** (*table*)
Table_PTR *table*;

Frees all memory associated with a *Table* object.

Table_PTR **BIF_copy_Table** (*table*)
Table_PTR *table*;

Duplicates a *Table* object and all memory associated with it.

Table_PTR **BIF_modify_Table** (*change,idents,entries,next,prev*)
Table_PTR *change*;
TableIdent_PTR *idents*;
Entry_PTR *entries*;
Table_PTR *next*;
Table_PTR *prev*;

Modifies any record in a *Table* object. Only non-null fields are considered.

Table_PTR **BIF_modify_CONCURRENT_Table** (*change,idents,entries,next,prev*)
Table_PTR *change*;
TableIdent_PTR *idents*;
ConcurrentEntry_PTR *entries*;
Table_PTR *next*;
Table_PTR *prev*;

Modifies any record in a *Table* object containing *Concurrent* object entries. Only non-null fields are considered.

int **BIF_query_Table_num_table_idents** (*table*)
Table_PTR *table*;

Returns the number of table identifiers in this table. Recall that a *TableIdent* object is a linked list, allowing complete definition of tables within hierarchies.

TableIdent_PTR **BIF_query_Table_table_ident** (*table*)

Table_PTR *table*;

Returns a *TableIdent* object representing the table name. The *TableIdent* object is a linked list and can be traversed from beginning to end to get a complete definition of this table's location in the hierarchy.

Bool **BIF_query_Table_OPSBASED** (*table*)
Table_PTR *table*;

Returns a boolean value indicating whether or not the table is a operations-based table. Currently, if the table is not concurrent than it is operations-based. Eventually, the data structure will be extended to allow unit-based tables.

Bool **BIF_query_Table_CONCURRENT** (*table*)
Table_PTR *table*;

Returns a boolean value indicating whether or not the table is a concurrent table.

int **BIF_query_Table_num_entries** (*table*)
Table_PTR *table*;

Returns the number of entries in the table. The table can be either concurrent or operations-based.

Entry_PTR **BIF_query_Table_entries** (*table*)
Table_PTR *table*;

Returns an *Entry* object containing the entries in the table.

ConcurrentEntry_PTR **BIF_query_Table_concurrent_entries** (*table*)
Table_PTR *table*;

Returns a *ConcurrentEntry* object containing the concurrent entries in the table.

Table_PTR **BIF_query_Table_next** (*table*)
Table_PTR *table*;

Returns the next *Table* object in a possible linked list of *Table* objects. The value 0 (Table_PTR 0) is returned if the end of the list has been reached.

Table_PTR **BIF_query_Table_prev** (*table*)
Table_PTR *table*;

Returns the previous *Table* object in a possible linked list of *Table* objects. The value 0 (Table_PTR 0) is returned if the beginning of the list has been reached.

File_PTR **BIF_add_Table_to_File** (*table,file*)
Table_PTR *table*;
File_PTR *file*;

Adds a *Table* object to a file. A pointer to the same file is returned. Since the *Table* object is a linked list, successive adds append the *Table* object to the end of the list. See *File* object description for more details.

2.21 File

A *File* object contains an entire BIF description of a design. It is called "File" because a complete BIF description is contained in one physical text file. Eventually, the BIF data structure routines may handle multiple file descriptions.

File_PTR **BIF_create_File** (*name,tables*)
char **name*;
Table_PTR *tables*;

Returns a *File* object having the name *name*, and containing state tables *tables*. *name* is copied and allocated.

void **BIF_free_File** (*file*)
File_PTR *file*;

Frees all memory associated with *File* object.

File_PTR **BIF_copy_File** (*file*)
File_PTR *file*;

Duplicates a *File* object and all memory associated with it.

File_PTR **BIF_modify_File** (*change,name,tables*)
File_PTR *change*;
char **name*;
Table_PTR *tables*;

Modifies any record in a *File* object. Only non-null fields are considered.

char ***BIF_query_File_name** (*file*)
File_PTR *file*;

Returns the character string name of the file.

int **BIF_query_File_num_tables** (*file*)
File_PTR *file*;

Returns the number of tables in the file.

Table_PTR **BIF_query_File_tables** (*file*)
File_PTR *file*;

Returns a *Table* object which contains the tables belonging to the *File* object.

2.22 Expression

The *Expression* object, and associated routines, provide a flexible way of creating and manipulating expressions. A wide variety of operators are supported, and precedence ordering is built-in. *Expression* objects, in combination with *Variable* objects, can represent expressions of arrays, events, and constants.

The *Expression* object is represented by a binary tree. The nodes in the tree are operators and leaves represent variables. Unary operators, as a special case, require only one operand. In this case the left child is ignored and the the right contains the operand.

Expr_PTR **BIF_create_Expression** (*op*,$expr_1$, $expr_2$, $expr_3$, ..., 0)
int *op*;
Expr_PTR $expr_1$,$expr_2$,$expr_3$, ...;

This routine creates an *Expression* object. The first parameter must be a valid operator. Operators are discussed below. Following the operator can be any number of *Expression* objects. The resulting expression tree represents the equivalent of: $expr_1$ op $expr_2$ op $expr_3$ op .... The list *must* be terminated with a 0 or NULL entry. For the case of unary operators, a single *Expression* object should follow the operator. Examples using this routine can be found in the appendix.

Expr_PTR **BIF_Variable_to_Expression** (*variable*)
Variable_PTR *variable*;

In order to use a variable in **BIF_create_Expression** the variable must first be converted to an *Expression* object, using this routine.

void **BIF_free_Expression** (*expression*)
Expr_PTR *expression*;

Frees all memory associated with an *Expresson* object. If the object represents an expression tree the tree is recursively freed.

Expr_PTR **BIF_copy_Expression** (*expression*)
Expr_PTR *expression*;

Copies an entire expression tree.

Expr_PTR **BIF_modify_LEAF_Expression** (*change*,*variable*)
Expr_PTR *change*;
Variable_PTR *variable*;

Modifies an *Expression* object that happens to be a leaf on the tree.

Expr_PTR **BIF_modify_NODE_Expression** (*change*,*operator*,*left*,*right*)
Expr_PTR *change*;
int *operator*;
Expr_PTR *left*;
Expr_PTR *right*;

Modifies an *Expression* object that happens to be a node on the tree. Operations are retrieved from the operator query routines described later on.

Bool **BIF_query_Expression_LEAF** (*expression*)
Expr_PTR *expression*;

Returns a boolean value indicating whether or not this expression is a leaf on the expression tree. If so, then the *Variable* object can be queried from it by using **BIF_query_Expression_variable**.

Bool **BIF_query_Expression_NODE** (*expression*)
Expr_PTR *expression*;

Returns a boolean value indicating whether or not this expression is a node in the expression tree. If so, the the operator can be queried from it by using **BIF_query_Expression_operator**.

Variable_PTR **BIF_query_Expression_variable** (*expression*)
Expr_PTR *expression*;

Returns the *Variable* object contained in the *Expression* object. If the expression is not a leaf then the value 0 (Variable_PTR 0) is returned.

int **BIF_query_Expression_operator** (*expression*)
Expr_PTR *expression*;

Returns the operator in the *Expression* object. If the expression is not a node then the value 0 (NOOP) is returned.

Expr_PTR **BIF_query_Expression_right** (*expression*)
Expr_PTR *expression*;

Returns the right child of the current *Expression* object. If the object is a leaf, and, therefore, has no children, the value 0 (Expr_PTR 0) is returned.

Expr_PTR **BIF_query_Expression_left** (*expression*)
Expr_PTR *expression*;

Returns the left child of the current *Expression* object. If the object is a leaf, and, therefore, has no children, the value 0 (Expr_PTR 0) is returned.

Bool **BIF_query_UNARY** (*op*)
int *op*;

Returns a boolean value indicating whether or not this operator is unary.

int **BIF_precedence** (*op1*,*op2*)
int *op1*;
int *op2*;

This function returns 0, if *op1* and *op2* have equal precedence, 1, if *op1* has greater precedence, and -1, if *op2* has greater precedence.

The following table shows all operator query functions. Each function returns an integer value that corresponds to the operator requested. This operator can than be used in **BIF_create_Expression**, or, compared with the operator returned from
BIF_query_Expression_operator to take some course of action.

char ***BIF_sprint_Expression** (*expression*)
Expr_PTR *expression*;

Operator Query Functions	
Function Name	*Operator Description*
BIF_query_OP_ADD()	Addition
BIF_query_OP_SUB()	Subtraction
BIF_query_OP_USUB()	Unary minus
BIF_query_OP_MUL()	Multiplication
BIF_query_OP_DIV()	Division
BIF_query_OP_MOD()	Modula
BIF_query_OP_LT()	Less than
BIF_query_OP_LTE()	Less than or equal
BIF_query_OP_GT()	Greater than
BIF_query_OP_GTE()	Greater than or equal
BIF_query_OP_EQ()	Is equal
BIF_query_OP_NEQ()	Is not equal
BIF_query_OP_SHL0()	Shift left. (zero in)
BIF_query_OP_SHL1()	Shift left. (one in)
BIF_query_OP_SHR0()	Shift right. (zero in)
BIF_query_OP_SHR1()	Shift left. (one in)
BIF_query_OP_ROTL()	Rotate left
BIF_query_OP_ROTR()	Rotate right
BIF_query_OP_AND()	Bitwise AND
BIF_query_OP_OR()	Bitwise OR
BIF_query_OP_XOR()	Bitwise XOR
BIF_query_OP_NOT()	Unary Bitwise Invert
BIF_query_OP_NAND()	Bitwise NAND
BIF_query_OP_NOR()	Bitwise NOR
BIF_query_OP_XNOR()	Bitwise XNOR
BIF_query_OP_LAND()	Logical AND
BIF_query_OP_LOR()	Logical OR
BIF_query_OP_LNOR()	Logical NOR
BIF_query_OP_LXOR()	Logical XOR
BIF_query_OP_LXNOR()	Logical XNOR
BIF_query_OP_CONCAT()	Concatenation
BIF_query_OP_NOOP()	No operator.

Figure 7: The Query Operator Types

This function returns an ascii string corresponding to the expression tree in the *Expression* object. The string has been allocated so it is suggested that the user free it after it is no longer useful.

char ***BIF_Variable_to_String** (*variable*)
Variable_PTR *variable*;

This function returns a string corresponding to the *Variable* object passed to it. NOTE: This function returns static internal storage. Do not alter or free the string returned.

3 References

References

[DuHG89] Dutt, N., Hadley, T., and Gajski, D., "BIF: A Behavioral Intermediate Format For High Level Synthesis," Tech. Rpt. # 89-03, UC Irvine, September, 1989.

[DuHG90] Dutt, N., Hadley, T., and Gajski, D., "An Intermediate Representation for Behavioral Synthesis," *27th Design Automation Conference*, June, 1990.

[HaCD90] Hadley, T., Cho, J. H., and Dutt, D., "Translating BIF into VHDL: Algorithms and Examples," Tech. Rpt. # 90-06, UC Irvine, June, 1990.

Index