Foreign Language Interface Manual
Including Support for ModelSim® DE/SE and Questa® SIM

Software Version 2020.4

Table of Contents

End-User License Agreement
with EDA Software Supplemental Terms

List of Figures

List of Tables

Chapter 1
Introduction

This chapter provides background for using the ModelSim/Questa FLI (Foreign Language Interface) for VHDL.

For a categorical listing of FLI functions, refer to FLI Functions by Category. For complete details on the functions including purpose, syntax, and usage, refer to FLI Function Definitions.

Note

For more complete information on current support for Questa SIM, refer to the Installation and Licensing Guide.

Using the VHDL FLI

FLI routines are C programming language functions that provide procedural access to information within the HDL simulator, executed via the vsim command. A user-written application can use these functions to traverse the hierarchy of an HDL design, get information about and set the values of VHDL objects in the design, get information about a simulation, and control (to some extent) a simulation run. The header file *mti.h* externs all of the FLI functions and types that can be used by an FLI application.

Note

The Tcl C interface is included in the FLI; you can find the *tcl.h* file in the *<install_dir>/ include* directory. Tk and Tix are not included in the FLI because the FLI is in the kernel, not the user interface.

Important Concepts

Before creating an FLI application, you should have a good understanding of the following concepts of simulation and foreign language operation.

- **Elaboration** — When the simulator starts, it first goes through an elaboration phase during which it loads and connects the entire design and sets initial values. During this phase it loads all foreign shared libraries and executes the initialization functions of all foreign architectures.

- **Simulation** — The simulation phase of the simulator begins when you execute the first run command and continues until you execute a quit or restart command. When you execute a restart command, the simulator goes through its elaboration phase again.

- **Foreign Architecture** — A foreign architecture is a design unit that is instantiated in a design but that does not (generally) contain any VHDL code. Instead it is a link to a C model that can communicate to the rest of the design through the ports of the foreign architecture. Normally, a C model creates processes and reads and drives signal values;

in essence, behaving in the same manner as VHDL code but with the advantage of the power of C and the ease of reading and writing files and communicating with other system processes.

- **Foreign Subprogram** — A foreign subprogram is a VHDL function or procedure that is implemented in C as opposed to VHDL. A foreign subprogram reads its in and inout parameters, performs some operation(s) which may include accessing simulator information through FLI function calls, writes its inout and out parameters, and returns a value (in the case of a function).

- **Callback** — A callback is a C function that is registered with the simulator for a specific reason. The simulator calls the registered function whenever the reason occurs. Callback functions generally perform special processing whenever certain simulation conditions occur.

- **Process** — A process is a VHDL process that is created through the FLI. It can either be scheduled for a specific time or be made sensitive to one or more signals that trigger the process to run. The process is associated with a C function and the C function is executed whenever the process is run by the simulator.

Using the VHDL FLI with Foreign Architectures

To use the foreign language interface with C models, you first create and compile an architecture with the FOREIGN attribute. The string value of the attribute is used to specify the name of a C initialization function and the name of an object file to load.

When the simulator elaborates the architecture, it calls the initialization function. Parameters to the function include a list of ports and a list of generics. Refer to Mapping to VHDL Data Types.

Declaring the FOREIGN Attribute

Starting with VHDL93, the FOREIGN attribute is declared in package STANDARD. With the 1987 version, you need to declare the attribute yourself. You can declare it in a separate package, or you can declare it directly in the architecture. (This will also work with VHDL93).

The FOREIGN Attribute String

The value of the FOREIGN attribute is a string containing three parts.

For the following declaration:

```
ATTRIBUTE foreign OF arch_name
: ARCHITECTURE IS "app_init app.so[; parameter]";
```

the attribute string parses this way:

- *app_init*— (required) The name of the initialization function for this architecture. Refer to The C Initialization Function.

- *app.so*— (required) The path to the shared object file to load. See The C Initialization Function.

- *parameter*— (optional) A string that is passed to the initialization function. This part is preceded by a semicolon.

If the you precede the initialization function with a plus (+) or minus (-), the simulator elaborates the VHDL architecture body in addition to the foreign code.

- + — (as in the example below), elaborate the VHDL first.

- - — elaborate the VHDL after the calling the foreign initialization function.

You can also use environment variables within the string, as in this example:

```
ATTRIBUTE foreign OF arch_name : ARCHITECTURE IS "+app_init $CAE/app.so";
```

Location of Shared Object Files

The simulator searches for object files in the following order:

1. *$MGC_WD/<so>* or *./<so>* (If MGC_WD is not set, then it will use ".")

2. <so>

3. within $LD_LIBRARY_PATH

4. *$MGC_HOME/lib/<so>*

5. *$MODEL_TECH/<so>*

6. *$MODEL_TECH/../<so>*

In the search information above, <so> refers to the shared library path specified in the FOREIGN attribute string. MGC_WD and MGC_HOME are user-definable environment variables. MODEL_TECH is set internally by vsim to the directory where the vsim executable resides.

_____ **Note** _____
The *.so* extension works on all platforms

The C Initialization Function

The initialization function is the entry point into the foreign C model. It typically does the following:

- Allocates memory to hold variables for the instance.

- Registers a callback function to free the memory when the simulator is restarted.

- Saves the handles to the signals in the port list.

- Creates drivers on the ports that will be driven.

- Creates one or more processes (a C function that can be called when a signal changes).

- Sensitizes each process to a list of signals.

The declaration of an initialization function is:

```
app_init(
    mtiRegionIdT        region,
    char                *param,
    mtiInterfaceListT   *generics,
    mtiInterfaceListT   *ports
)
```

The elaboration phase calls the function specified in the foreign attribute is.

- The first parameter is a region ID that determines the location in the design for this instance.

- The second parameter is the last part of the string in the foreign attribute.

- The third parameter is a linked list of the generic values for this instance. The list will be NULL if there are no generics.

- The last parameter is a linked list of the ports for this instance. The typedef mtiInterfaceListT in mti.h describes the entries in these lists.

Using the VHDL FLI with Foreign Subprograms

This section provides information on how you work with foreign subprograms.

Declaring a Foreign Subprogram in VHDL

To call a foreign C subprogram, you must write a VHDL subprogram declaration that has the equivalent VHDL parameters and return type. Then use the FOREIGN attribute to specify which C function and module to load.

The syntax of the FOREIGN attribute is almost identical to the syntax used for foreign architectures. For instance:

```
procedure in_params(
    vhdl_integer   : IN integer;
    vhdl_enum      : IN severity_level;
    vhdl_real      : IN real;
    vhdl_array     : IN string);

attribute FOREIGN of in_params : procedure is "in_params app.so";
```

You must also write a subprogram body for the subprogram, although it will never be called. For instance:

```
procedure in_params(
    vhdl_integer   : IN integer;
    vhdl_enum      : IN severity_level;
    vhdl_real      : IN real;
    vhdl_array     : IN string) is
begin
    report "ERROR: foreign subprogram in_params not called";
end;
```

Matching VHDL Parameters with C Parameters

You must accurately match the C parameters in your foreign C subprogram to the VHDL parameters in your VHDL package declaration.

The parameters must match in order as well as type.

Table 1-1. Parameters — VHDL to C Comparison

| VHDL Type | Parameters of class CONSTANT OR VARIABLE | | Parameters of class SIGNAL |
	IN	INOUT/OUT	IN
Integer	int	int *	mtiSignalIdT
Enumeration	int	char * if <= 256 values int * if > 256 values	mtiSignalIdT
Real	double *	double *	mtiSignalIdT
Time	mtiTime64T *	mtiTime64T *	mtiSignalIdT
Array	mtiVariableIdT	mtiVariableIdT	mtiVariableIdT
Record	mtiVariableIdT	mtiVariableIdT	mtiVariableIdT
File	Not supported		
Access Integer	int	int *	Not supported
Access Enumeration	int	int *	Not supported
Access Real	double *	double *	Not supported
Access Array	mtiVariableIdT *	mtiVariableIdT *	Not supported
Access File	Not supported		
Access Record	Not supported		

_____ **Note** _____

Handles to foreign subprogram parameters (non-signals) are not persistent. The handles are no longer valid after the subprogram has exited, so they cannot be saved and used later by other foreign code.

Arrays are not NULL-terminated. The length of an array can be determined by calling mti_TickLength() on the array's type.

Array and record SIGNAL parameters are passed as an mtiVariableIdT type. Any array or record sub-elements of these composite variables are also of type mtiVariableIdT. The values of all scalar sub-elements are of type mtiSignalIdT. To access the signal IDs, use mti_GetVarSubelements() at each composite level. For each sub-element that is of a scalar type, use mti_GetVarValueIndirect() to get the signal ID of the scalar. Once you have the signal IDs of the scalar sub-elements, you can use the FLI signal functions to manipulate the signals.

Matching VHDL Return Types with C Return Types

You must match the C return types in your foreign C subprogram to the VHDL return types in your VHDL code.

Table 1-2. Return Types — VHDL to C Comparison

VHDL Return Type	C Return Type
Integer	int
Enumeration	int
Real[1]	mtiRealT
Time[1]	mtiTime64T
Array	Not supported
File	Not supported
Record	Not supported
Access	Not supported

1. On Linux®, the compiler switch -freg-struct-return must be used when compiling any FLI application code that contains foreign functions that return real or time values.

MtiRealT is a special type that you must use as the return type of a foreign function that returns a real value. Macros are provided in *mti.h* for setting values in and getting values out of variables of type mtiRealT.

C Code and VHDL Examples

Use FOREIGN attribute specifications to connect C functions to VHDL procedures.

The following examples illustrate the association between C functions and VHDL procedures. The C function is connected to the VHDL procedure through the FOREIGN attribute specification.

C Subprogram Example

Functions declared in this code, in_params( ) and out_params( ), have parameters and return types that match the procedures in the subsequent package declaration (pkg).

```c
#include <stdio.h>
#include "mti.h"

char *severity[] = {"NOTE", "WARNING", "ERROR", "FAILURE"};
static char *get_string(mtiVariableIdT id);

void in_params (
        int             vhdl_integer,    /* IN integer          */
        int             vhdl_enum,       /* IN severity_level   */
        double          *vhdl_real,      /* IN real             */
        mtiVariableIdT  vhdl_array       /* IN string           */
        )
{
    printf("Integer   = %d\n", vhdl_integer);
    printf("Enum      = %s\n", severity[vhdl_enum]);
    printf("Real      = %g\n", *vhdl_real);
    printf("String    = %s\n", get_string(vhdl_array));
}

void out_params (
        int             *vhdl_integer,  /* OUT integer          */
        char            *vhdl_enum,     /* OUT severity_level   */
        double          *vhdl_real,     /* OUT real             */
        mtiVariableIdT  vhdl_array      /* OUT string           */
      )
{
    char *val;
    int i, len, first;

    *vhdl_integer += 1;

    *vhdl_enum += 1;
    if (*vhdl_enum > 3){
        *vhdl_enum = 0;
    }

    *vhdl_real += 1.01;

    /* rotate the array */
    val = mti_GetArrayVarValue(vhdl_array, NULL);
    len = mti_TickLength(mti_GetVarType(vhdl_array));
    first = val[0];
    for (i = 0; i < len - 1; i++){
        val[i] = val[i+1];
    }
    val[len - 1] = first;
}

/* Convert a VHDL String array into a NULL terminated string */
static char *get_string(mtiVariableIdT id)
{
    static char buf[1000];
    mtiTypeIdT type;
    int len;

    mti_GetArrayVarValue(id, buf);
    type = mti_GetVarType(id);
    len = mti_TickLength(type);
```

```
        buf[len] = 0;
        return buf;
    }
```

Package (pkg) Example

The FOREIGN attribute specification links the C functions (declared above) to VHDL
procedures (in_params( ) and out_params( )) in pkg.

```
package pkg is
    procedure in_params(
        vhdl_integer : IN integer;
        vhdl_enum    : IN severity_level;
        vhdl_real    : IN real;
        vhdl_array   : IN string);
    attribute foreign of in_params : procedure is "in_params test.sl";

    procedure out_params(
        vhdl_integer : OUT integer;
        vhdl_enum    : OUT severity_level;
        vhdl_real    : OUT real;
        vhdl_array   : OUT string);
    attribute foreign of out_params : procedure is "out_params test.sl";
end;

package body pkg is

    procedure in_params(
        vhdl_integer : IN integer;
        vhdl_enum    : IN severity_level;
        vhdl_real    : IN real;
        vhdl_array   : IN string) is
    begin
        report "ERROR: foreign subprogram in_params not called";
    end;

    procedure out_params(
        vhdl_integer : OUT integer;
        vhdl_enum    : OUT severity_level;
        vhdl_real    : OUT real;
        vhdl_array   : OUT string) is
    begin
        report "ERROR: foreign subprogram out_params not called";
    end;
end;
```

Entity (test) Example

The VHDL model *test* contains calls to procedures (in_params( ) and out_params( )) that are
declared in pkg and linked to functions in the C code.

```
    entity test is end test;
    use work.pkg.all;
    architecture only of test is
    begin
        process
            variable int : integer := 0;
            variable enum : severity_level := note;
            variable r    : real := 0.0;
            variable s    : string(1 to 5) := "abcde";
        begin
            for i in 1 to 10 loop
                in_params(int, enum, r, s);
                out_params(int, enum, r, s);
            end loop;
            wait;
        end process;
    end;
```

Mapping to VHDL Data Types

Many FLI functions have parameters and return values that represent VHDL object values. There are specific methods for mapping the object values to the various VHDL data types.

VHDL data types are identified in the C interface by a type ID. A type ID can be obtained for a signal by calling mti_GetSignalType( ) and for a variable by calling mti_GetVarType( ).

Alternatively, the mti_CreateScalarType( ), mti_CreateRealType( ), mti_CreateTimeType( ), mti_CreateEnumType( ), and mti_CreateArrayType( ) functions return type IDs for the data types they create.

Given a type ID handle, the mti_GetTypeKind( ) function returns a C enumeration of mtiTypeKindT that describes the data type. The mapping between mtiTypeKindT values and VHDL data types is as follows:

Table 1-3. Mapping to VHDL Data Types

mtiTypeKindT value	VHDL data type
MTI_TYPE_ACCESS	Access type (pointer)
MTI_TYPE_ARRAY	Array composite type
MTI_TYPE_C_ENUM	Enumeration scalar type for SystemC
MTI_TYPE_C_REAL	Floating point scalar type for SystemC
MTI_TYPE_ENUM	Enumeration scalar type
MTI_TYPE_FILE	File type
MTI_TYPE_PHYSICAL	Physical scalar type

Table 1-3. Mapping to VHDL Data Types (cont.)

mtiTypeKindT value	VHDL data type
MTI_TYPE_REAL	Floating point scalar type
MTI_TYPE_RECORD	Record composite type
MTI_TYPE_SCALAR	Integer scalar type
MTI_TYPE_TIME	Time type

The C interface does not support object values for access and file types. Values for record types are supported at the non-record subelement level. Effectively, this leaves scalar types and arrays of scalar types as valid types for C interface object values. In addition, multi-dimensional arrays are accessed in the same manner as arrays of arrays. For example, toto(x,y,z) is accessed as toto(x)(y)(z).

Scalar and physical types use 4 bytes of memory; TIME and REAL types use 8 bytes. An enumeration type uses either 1 byte or 4 bytes, depending on how many values are in the enumeration. If it has 256 or fewer values, then it uses 1 byte; otherwise, it uses 4 bytes. In some cases, all scalar types are cast to "long" before being passed as a non-array scalar object value across the C interface. The mti_GetSignalValue( ) function can be used to get the value of any non-array scalar signal object except TIME and REAL types, which can be retrieved using mti_GetSignalValueIndirect( ). Use mti_GetVarValue( ) and mti_GetVarValueIndirect( ) for variables.

Enumeration Types

Enumeration object values are equated to the position number of the corresponding identifier or character literal in the VHDL type declaration. For example:

```
-- C interface values
    TYPE std_ulogic IS('U',-- 0
                       'X',-- 1
                       '0',-- 2
                       '1',-- 3
                       'Z',-- 4
                       'W',-- 5
                       'L',-- 6
                       'H',-- 7
                       '-' -- 8
                         );
```

Real and Time Types

Eight bytes are required to store the values of variables and signals of type REAL and TIME. In C, this corresponds, respectively, to the C "double" data type and the mtiTime64T structure defined in *mti.h*. The mti_GetSignalValueIndirect( ) and mti_GetVarValueIndirect( ) functions are used to retrieve these values.

Array Types

The C type "void *" is used for array type object values. The pointer points to the first element of an array of C type "char" for enumeration types with 256 or fewer values, "double" for REAL types, "mtiTime64T" for TIME types, and "mtiInt32T" in all other cases. The first element of the array corresponds to the left bound of the array index range.

Multi-dimensional arrays are represented internally as arrays of arrays. For example, toto(x,y,z) is represented as toto(x)(y)(z). In order to get the values of the scalar subelements, you must use mti_GetSignalSubelements( ) or mti_GetVarSubelements( ) at each level of the array until you get to an array of scalars.

_____ **Note** _____

A STRING data type is represented as an array of enumeration values. The array is not NULL terminated as you would expect for a C string, so you must call mti_TickLength( ) to get its length.

Using Checkpoint/Restore with the FLI

In order to use checkpoint/restore with the FLI, any data structures that have been allocated in foreign models and certain IDs passed back from FLI function calls must be explicitly saved and restored.

There are two key features to aid with this process.

- Set of memory allocation functions — When you allocate memory with one of these functions, the simulator will automatically restore it for you to the same location in memory, ensuring that pointers into the memory will still be valid.

- Collection of explicit functions to save and restore data — You will need to use these functions for any pointers to your data structures and for IDs returned from FLI functions.

Pointers that you save and restore must point to memory allocated by the simulator. Objects in the shared library will no longer be valid if the shared library is reloaded into a different location during a restore. If you choose not to use the provided memory allocation functions, you will have to explicitly save and restore your allocated memory structures as well.

You must code your model assuming that the code could reside in a different memory location when restored. This requires that you update all process pointers during a restore and re-register all callback functions either in the init function or after the restore is complete.

Example

Example 1-1 shows a C model of a two-input AND gate taken from */<install_dir>/examples/ vhdl/foreign/example_two/gates.c*. It has been adapted for checkpoint/restore, and the added

lines are marked as comments. Note that this example addresses only one of the foreign architectures in the example file. If you plan to use the file from the installation directory, you should comment out the instantiation of the two foreign architectures that are not used.

Example 1-1. AND Gate with Checkpoint/Restore

```
--      dump: dump_design;
--      monit: monitor;
/*
 * This program creates a process sensitive to two signals and
 * whenever one or both of the signals change it does an AND operation
 * and drives the value onto a third signal.
 */

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

typedef struct {
    mtiSignalIdT in1;
    mtiSignalIdT in2;
    mtiDriverIdT out1;
    mtiProcessIdT proc; /* new */
} inst_rec;

void do_and( void * param )
{
    inst_rec * ip = (inst_rec *)param;
    mtiInt32T val1, val2;
    mtiInt32T result;

    val1  = mti_GetSignalValue( ip->in1 );
    val2  = mti_GetSignalValue( ip->in2 );
    result = val1 & val2;
    mti_ScheduleDriver( ip->out1, result, 0, MTI_INERTIAL );
}

void save_data( void * param ) /* new function */
{
    inst_rec * ip = (inst_rec *)param;

    mti_SaveBlock( (char*)&ip, sizeof(ip) );
}

void restore_data( void * param ) /* new function */
{
    inst_rec * ip;

    mti_RestoreBlock( (char*)&ip );
    mti_AddSaveCB( save_data, ip ); /* new */
    mti_RestoreProcess( ip->proc, "p1", do_and, ip );
}

void nop ( void * param )
{}

void and_gate_init(
    mtiRegionIdT       region,
    char              *param,
    mtiInterfaceListT *generics,
    mtiInterfaceListT *ports
)
```

```
{
    inst_rec *ip;
    mtiSignalIdT  outp;

    if ( ! mti_IsRestore() ) { /* new */
        ip = (inst_rec *)mti_Malloc( sizeof(inst_rec) );
        mti_AddSaveCB( save_data, ip ); /* new */
        ip->in1 = mti_FindPort( ports, "in1" );
        ip->in2 = mti_FindPort( ports, "in2" );
        outp = mti_FindPort( ports, "out1" );
        ip->out1 = mti_CreateDriver( outp );
        ip->proc = mti_CreateProcess( "p1", do_and, ip ); /* changed */
        mti_Sensitize( ip->proc, ip->in1, MTI_EVENT ); /* changed */
        mti_Sensitize( ip->proc, ip->in2, MTI_EVENT ); /* changed */ }
    else {
          mti_AddSaveCB( nop, NULL );  }

    mti_AddRestoreCB( restore_data, 0 ); /* new */ }
```

The above example displays the following features:

- mti_Malloc( ) is used instead of malloc( ).

- A callback is added using mti_AddRestoreCB( ) to restore the ip pointer and p1 process.

- Two callbacks are added using mti_AddSaveCB to save the ip pointer each time its value is determined.

- The mti_IsRestore( ) flag is checked for restore.

- When a restore is being done, mti_RestoreBlock( ) is used to restore the ip pointer because mti_SaveBlock( ) was used to save it. (Restores must be performed in the same order as saves.) The pointer is saved in this fashion because there are no mti_SavePointer/mti_RestorePointer routines.

- mti_RestoreProcess( ) is used to update the process created by mti_CreateProcess( ) with the possibly new address of the do_and( ) function. (This is in case the foreign code gets loaded into a different memory location.) All processes *must* be restored in this manner.

- All callbacks must be added with an mti_Add*() call during first initialization, restart, and restore. The restore does not restore callbacks because the routines might be located at different places after the restore operation.

The Effect of Restart on FLI Application Code

When issue a restart command to the simulator, it, by default, reloads shared libraries.

This reload is based on the conditions:

- Reloading a shared library previously loaded due to a foreign attribute on a VHDL architecture.

- Reloading a shared library previously loaded through the -foreign option to vsim.

You can override this default behavior in two ways.

- An FLI application can prevent reloading of the shared library in which it is contained by calling mti_KeepLoaded( ) during execution of its foreign architecture initialization function.

- Control reloading of all shared libraries with the vsim options -keeploaded and -keeploadedrestart, both of which prevent any shared libraries from being reloaded during a restart.

When the simulator reloads a shared library, the internal state of any FLI application which it contains is automatically reset to its initial state. But when a shared library is not reloaded, if any FLI application which it contains does not specifically check for a restart and reset its internal state to its initial state, then the internal state of that FLI application will remain in its last simulation state even though time has been reset to zero.

Because FLI shared libraries might or might not be reloaded during a restart, it is wise to always include a restart callback function (see mti_AddRestartCB()) in your FLI application that frees any memory that your code has allocated and resets the internal state of your application. It is also a good idea to avoid the use of static local variables.

Support for Verilog Instances

The FLI functions are designed to work with VHDL designs and VHDL objects. However, the functions for traversing the design hierarchy also recognize Verilog instances.

The following functions operate on Verilog instances as indicated:

- mti_GetTopRegion() — Gets the first top-level module. Use mti_NextRegion( ) to get additional top-level modules.

- mti_GetPrimaryName() — Gets the module name.

- mti_GetSecondaryName() — Returns NULL for normally compiled Verilog modules or the secondary name used for Verilog modules compiled with -fast.

The following functions operate on Verilog instances in the same manner as they operate on VHDL instances.

mti_CreateRegion()	mti_GetRegionFullName()
mti_FindRegion()	mti_GetRegionName()
mti_FirstLowerRegion()	mti_GetRegionSourceName()
mti_GetCurrentRegion()	mti_HigherRegion()
mti_GetLibraryName()	mti_NextRegion()

you cannot use functions that operate on VHDL signals and drivers on Verilog nets and drivers. For example, a call to mti_FirstSignal( ) on a Verilog region always returns NULL. You must use the PLI or VPI functions to operate on Verilog objects.

Support for SystemC

Many FLI functions can operate on SystemC objects.

For a complete list of which functions support SystemC, refer to FLI Functions by Category.

SystemC Regions

FLI functions that return an existing mtiRegionIdT can also return SystemC regions, include the following:

mti_FindRegion()	mti_HigherRegion()
mti_FirstLowerRegion()	mti_NextRegion()
mti_GetCallingRegion()	mti_GetProcessRegion()
mti_GetCurrentRegion()	mti_GetSignalRegion()
mti_GetTopRegion()	mti_GetRegionKind()

mti_CreateRegion() cannot create SystemC regions. It can, however, create an accForeign region within a SystemC parent region.

SystemC Signals

Many FLI accessor functions that return an existing mtiSignalIdT can return SystemC signals. These include the following:

mti_FindPort()

mti_FindSignal()

mti_FirstSignal()

mti_NextSignal()

mti_GetPhysicalData()

mti_GetSignalSubelements() can return individual bits of a SystemC vector. The function takes an mtiSignalIdT, which can be a SystemC vector (an error message results for a SystemC scalar or vector bit), and returns an array of mtiSignalIdTs, which can be SystemC vector bits.

The following functions, which all take an mtiSignalIdT, work correctly for SystemC signals:

mti_GetSignalValue()

mti_GetSignalValueIndirect()

mti_GetArraySignalValue()

Support for Windows Platforms

For the Windows operating system, sockets are separate objects from files and pipes, which require the use of different system calls.

There is no way to determine if a given descriptor is for a file or a socket. This necessitates the use of different callback functions for sockets under Windows. The following functions work specifically with sockets. While these functions are required for use with Windows, they are optional for use on Linux platforms.

- mti_AddSocketInputReadyCB()

- mti_AddSocketOutputReadyCB()

VHDL FLI Examples

Included in the ModelSim/Questa installation is a header file that must be included with foreign C code and several examples that illustrate how to use the foreign language interface.

The header file is:

> **/<install_dir>/include/mti.h**

The examples are located in:

> **/<install_dir>/examples/vhdl/foreign**

in the following directories:

- *example_one* — illustrates how to create processes and sensitize them to signals and how to read and drive signals from these processes.

- example_two — illustrates traversal of the design hierarchy, creation of a simple gate function, creation and sensitization of a process, and loading of multiple foreign shared libraries.

- *example_three* — illustrates how to read a testvector file and use it to stimulate and test a design via FLI function calls.

- *example_four* — illustrates how to create and use foreign subprograms.

Compiling and Linking FLI C Applications

The following platform-specific instructions show you how to compile and link your FLI applications so they can be loaded by ModelSim/Questa. Microsoft Visual C/C++ is supported

for creating Windows DLLs while gcc and cc compilers are supported for creating Linux shared libraries.

Although compilation and simulation switches are platform-specific, references to load shared objects are the same for all platforms. For information on loading objects see Using the VHDL FLI with Foreign Architectures and Declaring a Foreign Subprogram in VHDL.

Windows Platforms — Compiling C

Under Windows ModelSim/Questa loads a 32-bit dynamically linked library for each FLI application.

- Microsoft Visual C 4.1 or Later

```
cl -c -I<install_dir>\include
app.c link -dll -export:<C_init_function>
app.obj \      <install_dir>\win32\mtipli.lib
/out:app.dll
```

You can specify multiple -export options, one for each different FOREIGN attribute function name. The <C_init_function> argument is the function name specified in the FOREIGN attribute.

When executing cl commands in a DO file, use the /NOLOGO argument to prevent the Microsoft C compiler from writing the logo banner to stderr, which causes Tcl to think an error occurred.

If you need to run the Performance Analyzer on a design that contains FLI code, add these two arguments to the linking command shown above:

```
/DEBUG /DEBUGTYPE:COFF
```

which will add symbols to the .*dll* that the profiler can use in its report.

- MinGW gcc

```
gcc -c -I<install_dir>\include
app.c gcc -shared -o app.dll app.o -L<install_dir>\win32
-lmtipli
```

32-bit Linux Platform — Compiling C

If your foreign module uses anything from a system library, you will need to specify that library when you link your foreign module.

- gcc compiler:

```
gcc -c -I/<install_dir>/include
-m32 app.c gcc -shared -Wl, -Bsymbolic,--allow-shlib-undefined,--export-dynamic -o
app.so app.o -lc
```

When using -Bsymbolic, all symbols are first resolved within the shared library at link time. This will result in a list of undefined symbols. This is only a warning for shared

libraries and can be ignored. If you are using ModelSim/Questa on Redhat version 6.0 through 7.1, you also need to add the -noinhibit-exec argument when you specify -Bsymbolic.

The compiler switch -freg-struct-return must be used when compiling any FLI application code that contains foreign functions that return real or time values.

64-bit Linux for AMD64 and EM64T Platforms — Compiling C

64-bit Linux is supported on RedHat Enterprise Linux or SUSE Linux Enterprise Server.

- gcc compiler

    gcc -c -fPIC -I/<install_dir>/include
    app.c ld -shared -Bsymbolic -E --allow-shlib-undefined
    -o app.so app.o

To compile for 32-bit operation, specify the -m32 argument on the gcc command line.

If your FLI application requires a user or vendor-supplied C library, or an additional system library, you will need to specify that library when you link your FLI application. For example, to use the system math library libm, specify -lm to the ld command:

    gcc -c -fPIC -I/<install_dir>/include
    math_app.c ld -shared -Bsymbolic -E --allow-shlib-undefined
    \ -o math_app.so math_app.o -lm

Compiling and Linking FLI C++ Applications

The ModelSim/Questa simulator does not have direct support for any language other than standard C; however, you can load and execute C++ code under certain conditions.

Because the FLI functions have a standard C prototype, you must prevent the C++ compiler from mangling the FLI function names. This can be accomplished by using the following type of extern:

    extern "C"
    { <FLI application function prototypes>}

The header file *mti.h* already includes this type of extern. You must also put any functions specified in a VHDL foreign attribute inside of this type of extern.

You must also place an 'extern "C"' declaration immediately before the body of every import function in your C++ source code, for example:

```
extern "C"
int myimport(int i)
{
    vpi_printf("The value of i is %d\n", i); }
```

The following platform-specific instructions show you how to compile and link your FLI C++ applications so that they can be loaded by the simulator. Microsoft Visual C++ is supported for creating Windows DLLs while GNU C++ and native C++ compilers are supported for creating Linux shared libraries.

Although compilation and simulation switches are platform-specific, references to load shared libraries are the same for all platforms. For information on loading libraries, see Using the VHDL FLI with Foreign Architectures and Declaring a Foreign Subprogram in VHDL.

Windows Platforms

- Microsoft Visual C++ 4.1 or Later

  cl -c -GX -I<install_dir>\include
  app.cpp link -dll -export:<C_init_function>
  app.obj <install_dir>\win32\mtipli.lib

  The -GX option enables exception handling.

  You can specify multiple -export options, one for each different FOREIGN attribute function name, where <C_init_function> is the function name specified in the FOREIGN attribute.

  When executing cl commands in a DO file, use the /NOLOGO argument to prevent the Microsoft C++ compiler from writing the logo banner to stderr, which causes Tcl to think an error occurred.

  If you need to run the Performance Analyzer on a design that contains FLI code, add these two arguments to the linking command shown above:

  /DEBUG /DEBUGTYPE:COFF

  These arguments add symbols to the *.dll* that the profiler can use in its report.

- MinGWg++

  g++ -c -I<install_dir>\include
  app.cpp g++ -shared -o app.dll app.o -L<install_dir>\win32
  -lmtipli

32-bit Linux Platform

- GNU C++

  g++ -c -fPIC
  -I<install_dir>/modeltech/include app.cpp g++
  -shared -Bsymbolic -fPIC -o app.so app.o

64-bit Linux for AMD64 and EM64T Platforms

64-bit Linux®[1]is supported on RedHat Enterprise Linux or SUSE Linux Enterprise Server.

- GNU C++ compiler version gcc

g++ -c -fPIC -I/<install_dir>/include
app.cpp g++ -shared -Bsymbolic -o app.so app.o

If your FLI application requires a user or vendor-supplied C library, or an additional system library, you will need to specify that library when you link your FLI application. For example, to use the system math library libm, specify -lm:

g++ -c -fPIC -I/<install_dir>/include
math_app.cpp g++ -shared -Bsymbolic -o math_app.so
math_app.o -lm

---

1. Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

Using 64-bit with 32-bit FLI Applications

If you have 32-bit FLI applications and wish to use 64-bit ModelSim/Questa, you will need to port your code to 64 bits by moving from the ILP32 data model to the LP64 data model.

Porting C Code to the LP64 Data Model

There are several key points about porting C code to the LP64 data model.

- C long type grows to 64 bits.

- C pointer types, for example char*, grow to 64 bits.

- all other data types are unchanged from the 32-bit world.

- C long and pointer types do not fit in a C int.

- functions with no visible prototype are assumed to return int.

- C long and pointer values are truncated to 32 bits when:

  o   returned as an int type.

  o   assigned to an int type.

  o   cast to an int type.

  o   printf %d or %x formats are used instead of %ld or %lx.

- ints are zero- or sign-extended to 64 bits when

  o   returned as a long or pointer type.

  o   assigned to a long or pointer type.

  o   cast to a long or pointer type.

64-bit Time Values in the FLI

64-bit time values in the FLI are represented by the type mtiTime64T, which is defined in *mti.h*, where this type is defined as a 64-bit C long type on 64-bit systems, and as a C union type with 64-bit storage alignment on 32-bit systems.

C preprocessor macros are provided in *mti.h* to deal with mtiTime64T references and to make FLI code portable between 32- and 64-bit systems. The macros MTI_TIME64_INIT, MTI_TIME64_ASGN, MTI_TIME64_HI32, and MTI_TIME64_LO32 support initialization,

assignment, and reference to mtiTime64T objects as a 32-bit signed high-order component and a 32-bit unsigned low-order component. Here is a small example:

```
#include "mti.h"

mtiTime64T t1      = MTI_TIME64_INIT(0,1);    /* 1 */
mtiTime64T t2_32   = MTI_TIME64_INIT(1,0); /* 2**32 */
mtiTime64T tMinus1 = MTI_TIME64_INIT(-1,~0U); /* -1    */

void increment_time(mtiTime64T *tval)
{
#if defined(_LP64) || defined(__LP64__)
    *tval = *tval + 1;

#else
    int t_hi;
    unsigned int t_lo;
    t_hi = MTI_TIME64_HI32(*tval);
    t_lo = MTI_TIME64_LO32(*tval);
    ++t_lo;
    if (t_lo == 0)
        ++t_hi;
    MTI_TIME64_ASGN(*tval, t_hi, t_lo);
#endif
}
```

Shared Library Dependency

By default the simulator does not share the symbols defined by any of the shared libraries that it dynamically loads because of the many problems this can cause (such as, symbol clashing). However, you can load libraries with global symbol visibility by using the -gblso argument to vsim. Refer to the Command Reference Manual for details on this argument.

If you have a shared library that needs access to symbols in another shared library, but global visibility is not viable, then you must do one of the following:

- Put the shared library dependency in the link of the shared library that has the dependency.

- Code your shared library to dynamically load the shared library on which it depends and look up the symbols that it needs.

Example 1

This example shows two shared libraries, A and B. Library B uses functions in library A, and both libraries are dynamically loaded by ModelSim/Questa.

FLI Code

```
----- A.c -----

#include <mti.h>

int Afunc1( char * str )
{
    mti_PrintFormatted( "Afunc1 was called with parameter \"%s\".\n", str
);
    return 17;
}

int Afunc2( char * str )
{
    mti_PrintFormatted( "Afunc2 was called with parameter \"%s\".\n", str
);
    return 211;
}

void initForeign(
    mtiRegionIdT      region,
    char              *param,
    mtiInterfaceListT *generics,
    mtiInterfaceListT *ports
)
{
    mti_PrintFormatted( "+++ Shared lib A initialized.\n" );
}

----- B.c -----

#include <mti.h>

extern int Afunc1( char * );
extern int Afunc2( char * );

void loadDoneCallback( void * param )
{
    int retval;

    retval = Afunc2( "B calling Afunc2" );
    mti_PrintFormatted( "B called Afunc2 which returned %d.\n", retval );
}

void initForeign(
    mtiRegionIdT      region,
    char              *param,
    mtiInterfaceListT *generics,
    mtiInterfaceListT *ports
)
{
    int retval;

    mti_PrintFormatted( "+++ Shared lib B initialized.\n" );

    retval = Afunc1( "B calling Afunc1" );
    mti_PrintFormatted( "B called Afunc1 which returned %d.\n", retval );
```

```
        mti_AddLoadDoneCB( loadDoneCallback, 0 );
    }
```

Compilation Instructions

The following commands illustrate how to compile the files. Refer to Compiling and Linking FLI C Applications for instructions on other platforms.

> **gcc -c -I$MTI_HOME/include**
> **src/A.cld -G -Bsymbolic -o libA.so A.ogcc -c -I$MTI_HOME/include src/B.cld -G -Bsymbolic -L**
> **. -l A -o B.so B.o**

Simulation Output

```
setenv LD_LIBRARY_PATH .

vsim -c test -do test.do -foreign "initForeign libA.so" -foreign
"initForeign B.so"
Reading .../<product>/tcl/vsim/pref.tcl

# vsim -do test.do -foreign {initForeign libA.so} -foreign {initForeign
B.so} -c test
# Loading ../sunos5/../std.standard
# Loading work.test(a)
# Loading ./libA.so
# +++ Shared lib A initialized.
# Loading ./B.so
# +++ Shared lib B initialized.
# Afunc1 was called with parameter "B calling Afunc1".
# B called Afunc1 which returned 17.
# Afunc2 was called with parameter "B calling Afunc2".
# B called Afunc2 which returned 211.
# do test.do
```

Example 2

This example shows two shared libraries, A and C. ModelSim/Questa dynamically loads library C which in turn dynamically loads library A.

Library A is the same as in the previous example.

FLI Code

```
----- C.c -----

#include <dlfcn.h>
#include <mti.h>

typedef int (*funcPtrT)(char *);

void initForeign(
    mtiRegionIdT        region,
    char               *param,
    mtiInterfaceListT *generics,
    mtiInterfaceListT *ports
)
{
    int      retval;
    funcPtrT funcH;
    void *   libH;

mti_PrintFormatted( "+++ Shared lib C initialized.\n" );

    libH = dlopen("libA.so", RTLD_LAZY);
    if ( ! libH ) {
        mti_PrintMessage( "ERROR: Failed to load library libA.so.\n" );
    } else {
        funcH = dlsym( libH, "Afunc1" );
        if ( ! funcH ) {
            mti_PrintMessage( "ERROR: Failed to find function \"Afunc1\" "
                              "in library libA.so.\n" );
        } else {
            retval = funcH( "C calling Afunc1" );
            mti_PrintFormatted( "C called Afunc1 which returned %d.\n",
                                retval );
        }
    }
}
```

Compilation Instructions

The following commands illustrate how to compile the files. Refer to Compiling and Linking FLI C Applications for instructions on other platforms.

gcc -c -I\$MTI_HOME/include
src/A.cld -G -Bsymbolic -o libA.so A.ogcc -c -I\$MTI_HOME/include src/C.cld -G -Bsymbolic -o C.so C.o -ldl

Simulation output

```
setenv LD_LIBRARY_PATH .

vsim -c test -do test.do -foreign "initForeign C.so"
Reading .../<product>/tcl/vsim/pref.tcl

# vsim -do test.do -foreign {initForeign C.so} -c test
# Loading ../sunos5/../std.standard
# Loading work.test(a)
# Loading ./C.so
# +++ Shared lib C initialized.
# Afunc1 was called with parameter "C calling Afunc1".
# C called Afunc1 which returned 17.
# do test.do
```

------ **Note** ------

The initForeign() function in library A is not called by ModelSim/Questa because it does not load library A as a foreign library.

FLI Tracing

The foreign interface tracing feature is available for tracing user foreign language calls made to the MTI VHDL FLI. Foreign interface tracing creates two kinds of traces: a human-readable log of what functions were called, the value of the arguments, and the results returned; and a set of C-language files to replay what the foreign interface side did.

The Purpose of Tracing Files

The tracing logfile aids you in debugging FLI code.

The primary purpose of the replay facility is to send the replay file to support for debugging co-simulation problems, or debugging FLI problems for which it is impractical to send the FLI code. Support would still require a copy of the VHDL/Verilog part of the design to actually execute a replay, but many problems can be resolved with the trace only.

Invoking a Trace

To invoke the trace, execute **vsim** with the -trace_foreign option

Syntax

```
vsim -trace_foreign <action>
[-tag <name>]
```

Arguments

- <action>

    Specifies one of the following actions:

Value	Action	Result
1	create log only	writes a local file called "mti_trace_<tag>"
2	create replay only	writes local files called "mti_data_<tag>.c", "mti_init_<tag>.c", "mti_replay_<tag>.c" and "mti_top_<tag>.c"
3	create both log and replay	

- -tag <name>

(optional) Provides distinct file names for multiple traces.

Examples

- Create a logfile.

   vsim -trace_foreign 1 mydesign

- Create both a logfile and a set of replay files.

   vsim -trace_foreign 3 mydesign

- Create a logfile with a tag of "2".

   vsim -trace_foreign 1 -tag 2 mydesign

The tracing operations provide tracing during all user foreign code-calls, including VHDL foreign process callbacks and Verilog VCL callbacks. The miscellaneous VHDL callbacks (LoadComplete, Restart, Quit, EnvChanged, SimStatus, Save and Restore) are traced during execution but not explicitly identified as being from a callback function.

_____ **Note** _____
Tracing does not work across checkpoint/restore operations.

Debugging FLI Application Code

ModelSim/Questa offers the optional C Debug feature. This tool allows you to interactively debug C/C++ source code with the open-source gdb debugger.

Refer to the C Debug chapter in the User's Manual for details. If you do not have access to C Debug, continue reading for instructions on how to attach to an external C debugger.

In order to debug your FLI application code in a debugger, your application code must be compiled with debugging information (for example, by using the -g option argument). You must then load vsim into a debugger. Even though vsim is stripped, most debuggers will still execute it. You can invoke the debugger directly on vsimk, the simulation kernel where your application code is loaded (for example, "ddd `which vsimk`"), or you can attach the debugger to an already running vsim process. In the second case, you must attach to the PID for vsimk, and you must specify the full path to the vsimk executable (for example, "gdb $MTI_HOME/sunos5/vsimk 1234").

On Linux systems you can use either gdb or ddd.

Since initially the debugger recognizes only FLI function symbols from **vsim**, when invoking the debugger directly on vsim, you need to place a breakpoint in the first FLI function that is called by your application code. An easy way to set an entry point is to put a call to mti_GetProductVersion() as the first executable statement in your application code. Then, after

vsim has been loaded into the debugger, set a breakpoint in this function. Once you have set the breakpoint, run vsim with the usual arguments (for example, "run -c top").

When the breakpoint is reached, the shared library containing your application code has been loaded. In some debuggers you must use the share command to load the FLI application's symbols.

At this point all of the FLI application's symbols should be visible. You can now set breakpoints in and single step through your FLI application code.

FLI functions can be organized by category, as shown in this chapter.

For information on creating and using foreign architectures and subprograms, refer to the Chapter Introduction.

For complete details on the functions including purpose, syntax, and usage, refer to the Chapter FLI Function Definitions.

Function Categories

The FLI calls are organized into the following high-level categories.

Table 2-1. FLI Region Functions

Function[1]	Action
mti_CreateRegion()[2]	Creates a new region
mti_FindRegion()	Finds a region by name
mti_FirstLowerRegion()	Gets the first subregion inside of a region
mti_GetCallingRegion()	Gets the current elaboration region during elaboration or the region of the currently active process or signal resolution function or the current environment during simulation
mti_GetCurrentRegion()	Gets the current elaboration region during elaboration or the current environment during simulation
mti_GetGenericList()	Gets a list of the generics defined for a region
mti_GetTopRegion()	Gets the first top-level region
mti_HigherRegion()	Gets the parent region of a region
mti_NextRegion()	Gets the next region at the same level as a region
mti_GetLibraryName()	Gets the physical name of the library that contains a region
mti_GetPrimaryName()	Gets the primary name of a region (entity, package, or module)
mti_GetRegionFullName()	Gets the full hierarchical name of a region
mti_GetRegionKind()	Gets the type of a region (VHDL, Verilog, or SystemC)

Table 2-1. FLI Region Functions (cont.)

Function[1]	Action
mti_GetRegionName()	Gets the simple name of a region
mti_GetRegionSourceName()	Gets the name of the source file which contains a region
mti_GetSecondaryName()	Gets the secondary name of a region

1. All of these functions can access SystemC and Verilog objects.
2. The mti_CreateRegion() function cannot create SystemC or Verilog regions but it can create an accForeign region within a SystemC or Verilog parent region.

Table 2-2. FLI Process Functions

Function	Action
mti_CreateProcess()	Creates a new VHDL process
mti_CreateProcessWithPriority()	Creates a new VHDL process with a specific priority
mti_FirstProcess()[1]	Gets the first process in a region
mti_NextProcess()[1]	Gets the next process in a region
mti_GetProcessName()[1]	Gets the name of a process
mti_GetProcessRegion()[1]	Gets a handle to a process' region
mti_Sensitize()	Sensitizes a VHDL process to a VHDL signal
mti_Desensitize()	Desensitizes a VHDL process to the VHDL signals to which it is sensitive
mti_ScheduleWakeup()	Schedules a VHDL process to wake up at a specific time
mti_ScheduleWakeup64()	Schedules a VHDL process to wake up at a specific time using a 64-bit delay

1. This function can access SystemC objects.

Table 2-3. FLI Signal Functions

Function	Action
mti_CreateSignal()[1]	Creates a new VHDL signal
mti_FindPort()	Finds a port signal in a port interface list
mti_FindSignal()[2]	Finds signals by name
mti_FirstSignal()[2]	Gets the first signal in a region
mti_ForceSignal()	Forces a value onto a VHDL signal
mti_GetArraySignalValue()[2]	Gets the value of a signal of type array

Table 2-3. FLI Signal Functions (cont.)

Function	Action
mti_GetDrivingSignals()	Gets a handle to all of the signals driving a signal
mti_GetEquivSignal()	Finds the representation of the signal according to the simulator.
mti_GetParentSignal()[2]	Gets the higher up signal to which a signal is connected
mti_GetSignalMode()[2]	Gets the mode (direction) of a signal
mti_GetSignalName()[2]	Gets the simple name of a scalar or top-level composite signal
mti_GetSignalNameIndirect()[2]	Gets the full simple name of a signal including array indices and record subelement names
mti_GetSignalRegion()[2]	Gets the region in which a signal is declared
mti_GetSignalSubelements()[2]	Gets the subelements of a composite signal
mti_GetSignalType()[2]	Gets the type of a signal
mti_GetSignalValue()[2]	Gets the value of a VHDL or SystemC scalar signal of type enumeration, integer, or physical (VHDL only)
mti_GetSignalValueIndirect()[2]	Gets the value of a signal of any type except record
mti_NextSignal()[2]	Gets the next signal in a region
mti_ReleaseSignal()	Releases a force on a VHDL signal
mti_SetSignalValue()	Sets the value of a VHDL signal
mti_SignalImage()[2]	Gets the string image of a signal's value
mti_SignalIsResolved()	Indicates whether or not the specified signal is resolved

1. This function works with SystemC to the extent that you can create a VHDL signal in a SystemC region.
2. This function can access SystemC objects.

Table 2-4. FLI Driver Functions

Function	Action
mti_CreateDriver()	Creates a driver on a VHDL signal
mti_FindDriver()	Finds out if a VHDL signal has any drivers on it
mti_GetDriverNames()	Gets the names of all drivers on a VHDL signal
mti_GetDriverSubelements()	Gets the subelements of a composite driver
mti_GetDriverValues()	Gets the values of all drivers on a VHDL signal
mti_ScheduleDriver()	Schedules a driver to drive a value onto a VHDL signal

Table 2-4. FLI Driver Functions (cont.)

Function	Action
mti_ScheduleDriver64()	Schedules a driver to drive a value onto a VHDL signal with a 64-bit delay
mti_SetDriverOwner()	Sets the owning process of a driver

Table 2-5. FLI Variable Functions

Function	Action
mti_FindVar()	Finds a VHDL variable, generic, or constant by name
mti_FirstVar()	Gets the first VHDL variable, generic, or constant in a process
mti_NextVar()	Gets the next VHDL variable, generic, or constant in a process
mti_FirstVarByRegion()	Gets the first SystemC variable or constant visible in a region.
mti_GetArrayVarValue()	Gets the value of a VHDL variable of type array
mti_GetVarAddr()	Gets a pointer to a VHDL variable's value space
mti_GetVarImage()	Gets the string image of the value of a VHDL constant, generic, or variable (by name).
mti_GetVarImageById()	Gets the string image of a VHDL variable's value (by ID)
mti_GetVarKind()	Gets the kind of VHDL variable
mti_GetVarName()	Gets the simple name of a VHDL variable
mti_GetVarSubelements()	Gets the subelements of a composite VHDL variable
mti_GetVarType()	Gets the type of a VHDL variable
mti_GetVarValue()	Gets the value of a scalar VHDL variable of type enumeration, integer, or physical
mti_GetVarValueIndirect()	Gets the value of a VHDL variable of any type except record
mti_SetVarValue()	Sets the value of a VHDL variable

Table 2-6. FLI Type Functions

Function	Action
mti_CreateArrayType()	Creates an array type
mti_CreateEnumType()	Creates an enumeration type

Table 2-6. FLI Type Functions (cont.)

Function	Action
mti_CreateRealType()	Creates a real type
mti_CreateScalarType()	Creates a scalar type
mti_CreateTimeType()	Creates a time type
mti_GetArrayElementType()[1]	Gets the type of an array type's subelements
mti_GetNumRecordElements()	Gets the number of subelements in a record type
mti_GetEnumValues()[1]	Gets the values of an enumeration type
mti_GetPhysicalData()	Gets the unit information of a physical type
mti_GetTypeKind()[1]	Gets the kind of a type
mti_Image()[1]	Gets the string image of a value of a specific type
mti_IsSystemcSignedType()	Determines if the argument is a handle to a SystemC signed type.
mti_IsSystemcType()	Determines if the argument is a handle to a SystemC type.
mti_TickDir()[1]	Gets the direction of a type
mti_TickHigh()[1]	Gets the high value of a ranged type
mti_TickLeft()[1]	Gets the left value of a ranged type
mti_TickLength()[1]	Gets the length of a type
mti_TickLow()[1]	Gets the low value of a ranged type
mti_TickRight()[1]	Gets the right value of a ranged type

1. This function can access SystemC objects.

Table 2-7. FLI Callback Functions

Function	Action
mti_AddEnvCB()	Adds an environment change callback
mti_AddLoadDoneCB()	Adds an elaboration done callback
mti_AddQuitCB()	Adds a simulator exit callback
mti_AddRestartCB()	Adds a simulator restart callback
mti_AddRestoreCB()	Adds a simulator restore callback
mti_AddRestoreDoneCB()	Adds a simulator restore done callback

Table 2-7. FLI Callback Functions (cont.)

Function	Action
mti_AddSaveCB()	Adds a simulator checkpoint callback
mti_AddSimStatusCB()	Adds a simulator run status change callback
mti_AddInputReadyCB()	Adds or removes a file/pipe input ready callback
mti_AddOutputReadyCB()	Adds or removes a file/pipe output ready callback
mti_AddSocketInputReadyCB()	Adds or removes a socket input ready callback
mti_AddSocketOutputReadyCB()	Adds or removes a socket output ready callback
mti_AddUCDBSaveCB()	Adds the specified function to the coverage save callback list of the simulator.
mti_RemoveEnvCB()	Removes an environment change callback
mti_RemoveLoadDoneCB()	Removes an elaboration done callback
mti_RemoveQuitCB()	Removes a simulator exit callback
mti_RemoveRestartCB()	Removes a simulator restart callback
mti_RemoveRestoreCB()	Removes a simulator restore callback
mti_RemoveRestoreDoneCB()	Removes a simulator restore done callback
mti_RemoveSaveCB()	Removes a simulator checkpoint callback
mti_RemoveSimStatusCB()	Removes a simulator run status change callback
mti_AddDPISaveRestoreCB()	Adds a simulator checkpoint and restore callback for DPI.

Table 2-8. FLI Memory Management Functions

Function	Action
mti_Malloc()	Allocates simulator-managed memory
mti_Realloc()	Re-allocates simulator-managed memory
mti_Free()	Frees simulator-managed memory
mti_VsimFree()	Frees memory allocated by an FLI function that would normally be freed with the free() C-library function

Table 2-9. FLI Checkpoint/Restore Functions

Function	Action
mti_GetCheckpointFilename()	Gets the name of the current checkpoint file
mti_GetCheckpointDirname()	Returns the directory into which the checkpoint file is being written.

Table 2-9. FLI Checkpoint/Restore Functions (cont.)

Function	Action
mti_GetRestoreDirname()	Returns the name of the directory of the restore command.
mti_IsRestore()	Determines if a restore operation is in progress
mti_IsColdRestore()	Determines if a cold restore operation is in progress
mti_SaveBlock()	Saves a block of data to the checkpoint file
mti_SaveChar()	Saves a byte of data to the checkpoint file
mti_SaveLong()	Saves sizeof(long) bytes of data to the checkpoint file
mti_SaveShort()	Saves sizeof(short) bytes of data to the checkpoint file
mti_SaveString()	Saves a null-terminated string to the checkpoint file
mti_RestoreBlock()	Gets a block of data from the checkpoint file
mti_RestoreChar()	Gets a byte of data from the checkpoint file
mti_RestoreLong()	Gets sizeof(long) bytes of data from the checkpoint file
mti_RestoreShort()	Gets sizeof(short) bytes of data from the checkpoint file
mti_RestoreString()	Gets a null-terminated string from the checkpoint file
mti_RestoreProcess()	Restores a process that was created by mti_CreateProcess() or mti_CreateProcessWithPriority()

Table 2-10. FLI Time and Event Functions

Function	Action
mti_Delta()	Gets the simulator iteration count for the current time step
mti_Now()	Gets the low order 32 bits of the 64-bit current simulation time
mti_NowUpper()	Gets the high order 32 bits of the 64-bit current simulation time
mti_NowIndirect()	Gets the upper and lower 32 bits of the 64-bit current simulation time
mti_GetNextEventTime()	Gets the next event time (from a foreign subprogram or callback)
mti_GetNextNextEventTime()	Gets the next event time (from a VHDL process)
mti_GetResolutionLimit()	Gets the simulator resolution limit
mti_GetRunStopTime()	Gets the stop time of the current simulation run

Table 2-10. FLI Time and Event Functions (cont.)

Function	Action
mti_NowFormatted()	Returns the current simulation time formatted according to specified flags.
mti_TimeToString()	Returns a conversion of a specified time value, formatted according to specified flags.

Table 2-11. FLI Communication and Command Functions

Function	Action
mti_AddCommand()	Adds a user-defined simulator command
mti_AddTclCommand()	Adds a user-defined Tcl-style simulator command
mti_Exit()	Terminates the simulation immediately, returning the specified exit_status.
mti_Interp()	Gets the Tcl_Interp pointer used in the simulator
mti_Command()	Executes a simulator command
mti_Cmd()	Executes a simulator command with Tcl return status and no transcribing
mti_AskStdin()	Prompts the user for an input string
mti_PrintMessage()	Prints a message to the main transcript window
mti_PrintFormatted()	Prints a formatted message to the main transcript window
mti_Break()	Requests the simulator to halt
mti_FatalError()	Requests the simulator to halt with a fatal error
mti_Quit()	Requests the simulator to exit immediately
mti_QuitWithErrorCode()	Allows you to specify a nominal filename and line number along with an exit code, which is reported before terminating the simulator.

Table 2-12. FLI Miscellaneous Functions

Function	Action
mti_AddAttrToVsimTestrecord()	Adds an attribute to a test record of a UCDB database.
mti_GetAttrFromVsimTestrecord()	Returns an attribute from a test record of a UCDB database.
mti_RemoveAttrFromVsimTestrecord()	Removes an attribute from a test record of a UCDB database.
mti_GetProductVersion()	Gets the name and version of the simulator

Table 2-12. FLI Miscellaneous Functions (cont.)

Function	Action
mti_GetSimulationStatus()	Returns the current simulation status.
mti_GetWlfFilename()	Gets the name of the waveform logfile (.wlf)
mti_FindProjectEntry()	Gets the value of an entry in the project (.ini) file
mti_WriteProjectEntry()	Writes an entry to the project (.ini) file
mti_IsFirstInit()	Detects the first call to the initialization function
mti_KeepLoaded()	Requests that the current shared library not be unloaded on restart or load of a new design
mti_CallStack()	Prints a call stack from the point where the call is made

Chapter 3
FLI Function Definitions

This chapter describes the FLI functions in detail, explaining their purpose, syntax, and usage.

For information on creating and using foreign architectures and subprograms, refer to Introduction. For a categorical listing of FLI functions, refer to FLI Functions by Category.

Keep in mind the following caveats which are described further in the appropriate function descriptions:

- There are several FLI functions that work only during certain simulator phases (for example, mti_GetVarImage()), or only when called from a certain context (for example, from either inside of a process (mti_GetNextNextEventTime()) or outside of a process (mti_GetNextEventTime())).

- There are others that have slightly different behavior depending on when they are called and from which context (for example, mti_GetCurrentRegion() and mti_GetCallingRegion()).

- There are also several FLI functions that can be used on Verilog and SystemC regions in addition to VHDL regions (for example, mti_GetRegionKind()).

Function arguments are required unless marked as optional.

mti_AddAttrToVsimTestrecord()

Adds an attribute to a test record of a UCDB database.

Syntax

int = mti_AddAttrToVsimTestrecord(key, value)

Arguments

Name	Type	Description
key	const char *	Name of the attribute to be added.
value	void *	Value of the specified attribute.

Return Values

- 0 — upon successful addition of attribute

- -1 — otherwise

Description

Adds a new attribute on the vsim implicit test data record.

mti_AddCommand()

Adds a user-defined simulator command.

Syntax

mti_AddCommand(cmd_name, cmd_func)

Arguments

Name	Type	Description
cmd_name	char *	The name of the command.
cmd_func	mtiVoidFuncPtrT	A pointer to the function that will be called whenever the command is recognized by the command interpreter

Return Values

Nothing

Description

mti_AddCommand() adds the specified command to the simulator. The case of the command name is significant. The simulator command interpreter subsequently recognizes the command and calls the command function whenever the command is recognized. The entire command line (the command and any arguments) is passed to the command function as a character string. The command function prototype is:

```
void commandFuncName( char * command )
```

You can add a command with the same name as a previously added command (or even a standard simulator command), but only the last command you add has any effect.

Examples

FLI code

```
#include <mti.h>

void printSigInfo( void * param )
{
  char *        cp;
  char *        command = param;
  mtiSignalIdT sigid;
  mti_PrintFormatted( "Time [%d,%d] delta %d:\n", mti_NowUpper(),
                       mti_Now(), mti_Delta() );
  mti_PrintFormatted( "  Command: %s\n", command );
  for ( cp = command; (*cp != ' ') && (*cp != '\0'); cp++ ) { ; }
  for ( ; (*cp == ' ') && (*cp != '\0'); cp++ ) { ; }
  if ( *cp == '\0' ) {
      mti_PrintMessage( "    Usage: printSig <signame>\n" );
  } else {
      sigid = mti_FindSignal( cp );
      if ( ! sigid ) {
         mti_PrintFormatted( "    Signal %s not found.\n", cp );
      } else {
         switch ( mti_GetTypeKind( mti_GetSignalType( sigid ))) {
           case MTI_TYPE_SCALAR:
           case MTI_TYPE_ENUM:
           case MTI_TYPE_PHYSICAL:
             mti_PrintFormatted( "    Signal %s = %d\n", cp,
                                  mti_GetSignalValue( sigid ) );
            break;
          default:
             mti_PrintFormatted( "    The type of signal %s "
                                  "is not supported.\n", cp );
            break;
         }
      }
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this   */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the   */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign   */
                              /* model.                                */
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.*/
)
{
  mti_AddCommand( "printSig", printSigInfo );
}
```

HDL code

```
entity top is
end top;

architecture a of top is
  signal s1 : bit := '0';
  signal s2 : real := 1.0;
begin
  s1 <= not s1 after 5 ns;
  s2 <= s2 + 1.0 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> printSig
# Time [0,0] delta 0:
#  Command: printSig
#   Usage: printSig <signame>
VSIM 2> printSig /top/s2
# Time [0,0] delta 0:
#  Command: printSig /top/s2
#   The type of signal /top/s2 is not supported.
VSIM 3> printSig /top/s3
# Time [0,0] delta 0:
#  Command: printSig /top/s3
#   Signal /top/s3 not found.
VSIM 4> printSig /top/s1
# Time [0,0] delta 0:
#  Command: printSig /top/s1
#   Signal /top/s1 = 0
VSIM 5> run 5
VSIM 6> printSig /top/s1
# Time [0,5] delta 1:
#  Command: printSig /top/s1
#   Signal /top/s1 = 1
VSIM 7> quit
```

Related Topics

mti_AddSocketInputReadyCB()

mti_AddSocketOutputReadyCB()

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddDPISaveRestoreCB()

Adds a simulator checkpoint and restore callback for DPI.

Syntax

mti_AddDPISaveRestoreCB(saveFuncPtr, restoreFuncName)

Arguments

Name	Type	Description
saveFuncPtr	mtiVoidFuncPtrT	A pointer to a function to be called when the simulator does a checkpoint.
restoreFuncName	char*	The name of restore callback function. Must be non-NULL.

Return Values

Nothing

Description

mti_AddDPISaveRestoreCB() registers a pair of functions to the list of callback functions of the simulator's checkpoint and restore functionality.

The checkpoint callback function is registered via the function pointer, the matching restore callback function is registered via its name, and both functions take no parameters.

During a checkpoint operation, all checkpoint callbacks in the list are called in the same order of their registration. The callback function should save its state at this time. During restore, the restore functions will be resolved by their names and will be called in the same order of their registration. The restore function should restore its state at this time.

mti_AddDPISaveRestoreCB() can be called anywhere from DPI C code before a checkpoint command is issued.

Related Topics

mti_AddSocketInputReadyCB()

mti_AddSocketOutputReadyCB()

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddEnvCB()

Adds an environment change callback.

Syntax

mti_AddEnvCB(func, param)

Arguments

Name	Type	Description
func	mtiEnvCBFuncPtrT	A pointer to a function to be called whenever the simulation environment changes
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddEnvCB() adds the specified function to the simulator environment change callback list. The same function can be added multiple times, with possibly a different parameter each time. Whenever the simulator environment changes (for example, when the environment command is used), all callbacks in this list are called with their respective parameters plus a second parameter that is a pointer to the current context.

Examples

FLI code

```
#include <mti.h>
void envCallback( void * param, void * context )
{
  mtiRegionIdT region = (mtiRegionIdT)param;

  mti_PrintFormatted( "Foreign Arch in Region %s: "
                      "the current region is now %s.\n",
                      mti_GetRegionName( region ),
                      mti_GetRegionName( context ) );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddEnvCB( envCallback, region );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity bottom is
end bottom;

architecture b of bottom is
begin
end b;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component bottom is
  end component;

begin

  bot : bottom;

  i1 : for_model;

  s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.bottom(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Foreign Arch in Region i1: the current region is now top.
VSIM 1> env /top
# sim:/top
VSIM 2> env /top/i1
# Foreign Arch in Region i1: the current region is now i1.
# sim:/top/i1
VSIM 3> env /top/bot
# Foreign Arch in Region i1: the current region is now bot.
# sim:/top/bot
VSIM 4> env /top
# Foreign Arch in Region i1: the current region is now top.
# sim:/top
VSIM 5> quit
```

Related Topics

mti_AddSocketInputReadyCB()

mti_AddSocketOutputReadyCB()

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddInputReadyCB()

Add or remove a file/pipe(/socket) input ready callback.

Syntax

mti_AddInputReadyCB(file_desc, func, param)

Arguments

Name	Type	Description
file_desc	int	On Linux, a file, pipe, or socket descriptor
		On Windows, a pipe descriptor
func	mtiVoidFuncPtrT	A pointer to a function to be called whenever there is data available for reading on the file descriptor
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddInputReadyCB() puts a watch on the specified file descriptor. Whenever the file descriptor has data available for reading, the specified function is called along with its parameter.

In a Linux environment, mti_AddInputReadyCB() can be used with files, pipes, and sockets. In a Windows environment, it can be used only with pipes. (See mti_AddSocketInputReadyCB().)

To remove a previously added callback, call mti_AddInputReadyCB() with the same file descriptor but with a NULL function pointer.

Examples

FLI code

```c
#include <mti.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <stdlib.h>

#ifdef WIN32
#include <winsock.h>
#else
#include <unistd.h>
#include <sys/time.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>    /* gethostbyname() */
#endif

#ifdef HP700
#include <resolv.h>
#endif

#ifdef WIN32
#define MAXHOSTNAMELEN MAXGETHOSTSTRUCT
#else
#define SOCKET_ERROR -1
#define INVALID_SOCKET -1
typedef int SOCKET;
#endif

void sockCB( void * sock )
{
  int  i;
  char buf[1];

#ifdef WIN32
  i = recv( (SOCKET)sock, buf, sizeof(buf), 0 );
#else
  i = read( (SOCKET)sock, buf, sizeof(buf) );
#endif
  mti_PrintFormatted( "Read returned %d - Read %c\n", i, buf[0] );

  if ( (i == 0) || (buf[0] == 'C') ) {
      /* Remove the callback. */
#ifdef WIN32
      mti_AddSocketInputReadyCB( (SOCKET)sock, (mtiVoidFuncPtrT)0, 0 );
#else
      mti_AddInputReadyCB( (SOCKET)sock, (mtiVoidFuncPtrT)0, 0 );
#endif
      mti_PrintMessage("Closing socket\n");
      close( (SOCKET)sock );
  }
}

void loadDoneCB( void * sock )
```

```
{
  mti_PrintMessage( "Load Done: Adding socket callback.\n" );
#ifdef WIN32
  mti_AddSocketInputReadyCB( (SOCKET)sock, (mtiVoidFuncPtrT)sockCB, sock );
#else
  mti_AddInputReadyCB( (SOCKET)sock, (mtiVoidFuncPtrT)sockCB, sock );
#endif
}

void initForeign(
  mtiRegionIdT       region,    /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
  char               *param,    /* The last part of the string in the    */
                                /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  char   hostname[MAXHOSTNAMELEN] = "localhost";
  int    statusFlags;
  int    server_length;
  int    status;
  int    retry_cnt = 0;
  short  portNum = 0;
  struct sockaddr_in server;
  struct hostent *hp;
  SOCKET sock = INVALID_SOCKET;
#ifdef WIN32
  WORD    wVersionRequested;
  WSADATA wsaData;
  int     err;

  wVersionRequested = MAKEWORD( 1, 1 );
  err = WSAStartup( wVersionRequested, &wsaData );

  if ( err != 0 ) {
     mti_PrintMessage( "Cannot find a usable winsock.dll.\n" );
     return;
  }

/* Confirm that the Windows Sockets DLL supports 1.1.  Note that if
 * the DLL supports versions greater than 1.1 in addition to 1.1,
 * it will still return 1.1 in wVersion since that is the version
 * we requested.
 */

  if ( (LOBYTE( wsaData.wVersion ) != 1) ||
       (HIBYTE( wsaData.wVersion ) != 1) ) {
     mti_PrintMessage( "Cannot find a usable winsock.dll.\n" );
     WSACleanup();
     return;
  }
/* The Windows Sockets DLL is acceptable. Proceed. */

#endif

  sock = socket( AF_INET, SOCK_STREAM, 0 );
```

```
    if ( sock == INVALID_SOCKET ) {
#ifdef WIN32
        DWORD le = GetLastError();
        mti_PrintFormatted( "Error opening socket. Error=%d\n", le );
#else
        mti_PrintMessage( "Error opening socket.\n" );
#endif
        return;
    }

    while ( retry_cnt < 2 ) {
        memset( (char *)&server, 0, sizeof(server) );
        server.sin_family = AF_INET;
        if ( (hp = gethostbyname(hostname)) == 0 ) {
            mti_PrintFormatted( "%s: Unknown host.\n", hostname );
            close( sock );
            return;
        }
        memcpy( (char *)&server.sin_addr, (char *)hp->h_addr, hp->h_length );
        portNum        = 19; /* 'chargen' */
        server.sin_port = htons(portNum);
        server_length   = sizeof(server);
        status = connect( sock, (struct sockaddr *)&server, server_length );
        if ( status < 0 ) {
            if ( retry_cnt++ > 1 ) {
                 mti_PrintFormatted( "Error connecting to server %s:%d\n",
                                     hostname, portNum );
                close( sock );
            } else {
                strcpy( hostname, "map" );  /* Put your hostname here. */
            }
        }
    }

#ifdef WIN32
    {
        unsigned long non_blocking = 1;
        status = ioctlsocket( sock, FIONBIO, &non_blocking );
        if ( status == SOCKET_ERROR ) {
            perror( "Setting socket status" );
        }
    }
#else
    statusFlags = fcntl( sock, F_GETFL );
    if ( statusFlags == SOCKET_ERROR ) {
        perror( "Getting socket status" );
    } else {
        int ctlValue;
        statusFlags |= O_NONBLOCK;
        ctlValue = fcntl( sock, F_SETFL, statusFlags );
        if ( ctlValue == SOCKET_ERROR ) {
            perror( "Setting socket status" );
        }
    }
#endif

    mti_AddLoadDoneCB( (mtiVoidFuncPtrT)loadDoneCB, (void *)sock );
}
```

HDL code

```
entity top is
end top;

architecture a of top is
   signal s1 : bit := '0';
begin
   s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c -foreign "initForeign ./for_model.sl" top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign ./for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
# Load Done: Adding socket callback.
# Read returned 1 - Read
# Read returned 1 - Read !
# Read returned 1 - Read "
# Read returned 1 - Read #
# Read returned 1 - Read $
# Read returned 1 - Read %
# Read returned 1 - Read &
# Read returned 1 - Read '
# Read returned 1 - Read (
# Read returned 1 - Read )
# Read returned 1 - Read *
# Read returned 1 - Read +
# Read returned 1 - Read ,
# Read returned 1 - Read -
# Read returned 1 - Read .
# Read returned 1 - Read /
# Read returned 1 - Read 0
# Read returned 1 - Read 1
# Read returned 1 - Read 2
# Read returned 1 - Read 3
# Read returned 1 - Read 4
# Read returned 1 - Read 5
# Read returned 1 - Read 6
# Read returned 1 - Read 7
# Read returned 1 - Read 8
# Read returned 1 - Read 9
# Read returned 1 - Read :
# Read returned 1 - Read ;
# Read returned 1 - Read <
# Read returned 1 - Read =
# Read returned 1 - Read >
# Read returned 1 - Read ?
# Read returned 1 - Read @
# Read returned 1 - Read A
# Read returned 1 - Read B
# Read returned 1 - Read C
# Closing socket
VSIM 1> quit
```

Related Topics

mti_AddSocketOutputReadyCB()

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddLoadDoneCB()

Adds an elaboration done callback.

Syntax

mti_AddLoadDoneCB(func, param);

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function to be called at the end of elaboration
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddLoadDoneCB() adds the specified function to the elaboration done callback list. You can add the same function multiple times, with possibly a different parameter each time. At the end of elaboration, all callbacks in the list are called with their respective parameters. These callbacks are also called at the end of a restart or a cold restore (vsim -restore).

You must call mti_AddLoadDoneCB() from a foreign initialization function in order for the callback to take effect. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

Examples

FLI code

```
#include <mti.h>

void loadDoneCallback( void * param )
{
  mtiRegionIdT region = (mtiRegionIdT)param;

  mti_PrintFormatted( "Foreign Arch in Region %s: "
                      "the top-level region is %s.\n",
                      mti_GetRegionName( region ),
                      mti_GetRegionName( mti_GetTopRegion() ) );
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this   */
                                /* foreign architecture is instantiated.  */
  char               *param,    /* The last part of the string in the    */
                                /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCallback, region );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity bottom is
end bottom;

architecture b of bottom is
begin
end b;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component bottom is
  end component;

begin

  bot : bottom;

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.bottom(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Foreign Arch in Region i1: the top-level region is top.
VSIM 1> quit
```

Related Topics

mti_AddSocketOutputReadyCB()

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddOutputReadyCB()

Adds or removes a file/pipe(/socket) output ready callback.

Syntax

mti_AddOutputReadyCB(file_desc, func, param)

Arguments

Name	Type	Description
file_desc	int	On Linux, a file, pipe, or socket descriptor
		On Windows, a pipe descriptor
func	mtiVoidFuncPtrT	A pointer to a function to be called whenever the file descriptor is available for writing
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddOutputReadyCB() puts a watch on the specified file descriptor. Whenever the file descriptor is available for writing, the specified function is called along with its parameter.

In a Linux environment, you can use mti_AddOutputReadyCB() with files, pipes, and sockets; in a Windows environment, only with pipes. (See mti_AddSocketOutputReadyCB().)

To remove a previously added callback, call mti_AddOutputReadyCB() with the same file descriptor but with a NULL function pointer.

Related Topics

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddQuitCB()

Adds a simulator exit callback.

Syntax

mti_AddQuitCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function to be called when the simulator exits
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddQuitCB() adds the specified function to the simulator exit callback list. You can add the same function multiple times, with possibly a different parameter each time. When the simulator exits, it calls all callbacks in the list with their respective parameters.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

void quitCallback( void * param )
{
  mti_PrintFormatted( "Cleaning up %s for simulator exit ...\n",
                      (char *)param );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  char * instance_info;

  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddQuitCB( quitCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> quit
# Cleaning up for_model for simulator exit ...
```

Related Topics

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddRestartCB()

Adds a simulator restart callback.

Syntax

mti_AddRestartCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function to be called when the simulator restarts
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddRestartCB() adds the specified function to the simulator restart callback list. You can add the same function multiple times, with possibly a different parameter each time. When the simulator restarts, it calls all callbacks in the list with their respective parameters before the simulator is restarted. The callback function should do a cleanup operation including freeing any allocated memory and resetting global/static variables.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

void restartCallback( void * param )
{
  mti_PrintFormatted( "Cleaning up %s for simulator restart ...\n",
                       (char *)param );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  char * instance_info;

  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddRestartCB( restartCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 10
VSIM 2> restart -f
# Cleaning up for_model for simulator restart ...
# Loading ./for_model.sl
VSIM 3> quit
```

Related Topics

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddRestoreCB()

Adds a simulator restore callback.

Syntax

mti_AddRestoreCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function to be called when the simulator does a restore
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddRestoreCB() adds the specified function to the simulator restore callback list. You can add the same function multiple times, with possibly a different parameter each time. During a restore, the simulator calls all callbacks in the list with their respective parameters. The callback function should restore its saved state at this time.

You must call mti_AddRestoreCB() from a foreign initialization function in order for the callback to take effect. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cp.file
# Saving instance info "for_model"
VSIM 3> run 30
VSIM 4> restore cp.file
# Loading checkpoint/restore data from file "cp.file"
# Checkpoint created Thu Apr 27 15:52:32 2000
# Restoring state at time 20 ns, iteration 1
# Restored instance info "for_model"
VSIM 5> run 10
VSIM 6> quit
# Cleaning up...
```

Related Topics

mti_AddLoadDoneCB()

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mti_AddRestoreDoneCB()

Adds a simulator warm restore done callback.

Syntax

mti_AddRestoreDoneCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function to be called after the simulator completes a warm restore
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddRestoreDoneCB() adds the specified function to the simulator warm restore done callback list. You can add the same function multiple times, with possibly a different parameter each time. After the simulator completes a warm restore but before it returns control to the user, it calls all callbacks in the list with their respective parameters.

You must call mti_AddRestoreDoneCB() from a foreign initialization function in order for the callback to take effect. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

For cold restores (that is, vsim -restore), the simulator does not call restore-done callbacks at the end of the restore process. Instead it calls the load-done callbacks (see mti_AddLoadDoneCB()).

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void restoreDoneCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "\"%s\": Restore complete\n", inst_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddRestoreDoneCB( restoreDoneCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is
  signal s1 : bit := '0';
  component for_model is
  end component;
  for all : for_model use entity work.for_model(a);

begin
  i1 : for_model;
  s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cp.file
# Saving instance info "for_model"
VSIM 3> run 30
VSIM 4> restore cp.file
# Loading checkpoint/restore data from file "cp.file"
# Checkpoint created Thu Apr 27 15:52:32 2000
# Restoring state at time 20 ns, iteration 1
# Restored instance info "for_model"
# "for_model": Restore complete
VSIM 5> run 10
VSIM 6> quit
# Cleaning up...
```

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddSaveCB()

Adds a simulator checkpoint callback.

Syntax

mti_AddSaveCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function to be called when the simulator does a checkpoint
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddSaveCB() adds the specified function to the simulator checkpoint callback list. You can add the same function multiple times, with possibly a different parameter each time. During a checkpoint operation, the simulator calls all callbacks in the list with their respective parameters. The callback function should save its state at this time.

You should call mti_AddSaveCB() from a foreign initialization function. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cp.file
# Saving instance info "for_model"
VSIM 3> run 30
VSIM 4> restore cp.file
# Loading checkpoint/restore data from file "cp.file"
# Checkpoint created Thu Apr 27 15:52:32 2000
# Restoring state at time 20 ns, iteration 1
# Restored instance info "for_model"
VSIM 5> run 10
VSIM 6> quit
# Cleaning up...
```

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mti_AddSimStatusCB()

Adds a simulator run status change callback.

Syntax

mti_AddSimStatusCB(func, param)

Arguments

Name	Type	Description
func	mtiSimStatusCBFuncPtr T	A pointer to a function to be called whenever the simulator run status changes
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddSimStatusCB() adds the specified function to the simulator run status change callback list. You can add the same function multiple times, with possibly a different parameter each time. Whenever the simulator run status changes, it calls all callbacks in the list with their respective parameters plus a second parameter of type int which is 1 when the simulator is about to start a run and 0 when the run completes.

Examples

FLI code

```
#include <mti.h>

void simStatusCallback( void * param, int run_status )
{
  mtiRegionIdT region = (mtiRegionIdT)param;

  mti_PrintFormatted( "Time [%d,%d]: Region %s: the simulator %s\n",
                      mti_NowUpper(), mti_Now(),
                      mti_GetRegionName( region ),
                      (run_status == 1) ? "is about to run" :
                                          "just completed a run" );
}

void initForeign(
  mtiRegionIdT        region,    /* The ID of the region in which this   */
                                 /* foreign architecture is instantiated. */
  char              *param,      /* The last part of the string in the    */
                                 /* foreign attribute.                    */
  mtiInterfaceListT *generics,  /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports      /* A list of ports for the foreign model.  */
)
{
  mti_AddSimStatusCB( simStatusCallback, region );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 10
# Time [0,0]: Region i1: the simulator is about to run
# Time [0,10]: Region i1: the simulator just completed a run
VSIM 2> run 15
# Time [0,10]: Region i1: the simulator is about to run
# Time [0,25]: Region i1: the simulator just completed a run
VSIM 3> quit
```

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddSocketInputReadyCB()

Adds or removes a socket input ready callback.

Syntax

mti_AddSocketInputReadyCB(socket_desc, func, param)

Arguments

Name	Type	Description
socket_desc	int	A socket descriptor
func	mtiVoidFuncPtrT	A pointer to a function to be called whenever there is data available for reading on the socket descriptor
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddSocketInputReadyCB() puts a watch on the specified socket descriptor. Whenever the socket descriptor has data available for reading, the simulator calls the specified function along with its parameter.

To remove a previously added callback, call mti_AddSocketInputReadyCB() with the same socket descriptor but with a NULL function pointer.

mti_AddSocketInputReadyCB() and mti_AddSocketOutputReadyCB() are useful in setting up cosimulation environments where FLI code uses sockets to communicate with other processes. In the course of initialization, a cosimulation application typically would use standard system library routines to create or open a socket and obtain a socket descriptor and then call mti_AddSocketInputReadyCB() and mti_AddSocketOutputReadyCB() to set up the callback functions. During simulation, FLI code may initiate a non-blocking I/O operation on the socket (again using standard system library routines) and immediately return control to the simulator. When the I/O is completed, the simulator invokes the callback function which could check for errors, handle received data, or initiate another non-blocking I/O operation before returning to the simulator.

Examples

FLI code

```
#include <mti.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <stdlib.h>
#ifdef WIN32
#include <winsock.h>
#else
#include <unistd.h>
#include <sys/time.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>    /* gethostbyname()
*/
#endif
#ifdef HP700
#include <resolv.h>
#endif
#ifdef WIN32
#define MAXHOSTNAMELEN MAXGETHOSTSTRUCT
#else
#define SOCKET_ERROR -1
#define INVALID_SOCKET -1
typedef int SOCKET;#endif
void sockCB( void * sock )
{
  int  i;
  char buf[1];
#ifdef WIN32
  i = recv( (SOCKET)sock, buf, sizeof(buf),
0 );
#else
  i = read( (SOCKET)sock, buf, sizeof(buf)
);
#endif
  mti_PrintFormatted( "Read returned %d -
Read %c\n", i, buf[0] );
  if ( (i == 0) || (buf[0] == 'C') ) {
      /* Remove the callback. */
      mti_AddSocketInputReadyCB( (SOCKET)sock,
(mtiVoidFuncPtrT)0, 0 );        mti_PrintMessage("Closing socket\n");
      close( (SOCKET)sock );
   }
}
void loadDoneCB( void * sock )
{
  mti_PrintMessage( "Load Done: Adding socket
callback.\n" );
  mti_AddSocketInputReadyCB( (SOCKET)sock,
(mtiVoidFuncPtrT)sockCB, sock );
}
void initForeign(
  mtiRegionIdT       region,   /* The ID
```

```
                               of the region in which this      */
                                                    /* foreign
architecture is instantiated.    */
  char                *param,    /* The last
part of the string in the      */
                                                    /* foreign
attribute.                             */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.    */
)
{
  char    hostname[MAXHOSTNAMELEN] = "localhost";
  int     statusFlags;
  int     server_length;
  int     status;
  int     retry_cnt = 0;
  short   portNum = 0;
  struct sockaddr_in server;
  struct hostent *hp;
  SOCKET sock = INVALID_SOCKET;
  #ifdef WIN32
  WORD     wVersionRequested;
  WSADATA wsaData;
  int      err;
  wVersionRequested = MAKEWORD( 1, 1 );
  err = WSAStartup( wVersionRequested, &wsaData
);
  if ( err != 0 ) {
    mti_PrintMessage( "Cannot find a usable
winsock.dll.\n" );
    return;
  }
/* Confirm that the Windows Sockets DLL supports
1.1.  Note that if
 * the DLL supports versions greater than
1.1 in addition to 1.1,
 * it will still return 1.1 in wVersion since
that is the version
 * we requested.
 */
  if ( (LOBYTE( wsaData.wVersion ) != 1)
||
      (HIBYTE( wsaData.wVersion ) != 1)
) {
      mti_PrintMessage( "Cannot find a usable
winsock.dll.\n" );
      WSACleanup();
      return;
  }
/* The Windows Sockets DLL is acceptable.
Proceed. */
#endif
  sock = socket( AF_INET, SOCK_STREAM, 0
);
  if ( sock == INVALID_SOCKET ) {
#ifdef WIN32
```

```
        DWORD le = GetLastError();
        mti_PrintFormatted( "Error opening socket.
Error=%d\n", le );
#else
        mti_PrintMessage( "Error opening socket.\n"
);
#endif
        return;
    }
    while ( retry_cnt < 2 ) {
        memset( (char *)&server, 0, sizeof(server)
);
        server.sin_family = AF_INET;
        if ( (hp = gethostbyname(hostname)) ==
0 ) {
            mti_PrintFormatted( "%s: Unknown host.\n",
hostname );
            close( sock );
            return;
        }
        memcpy( (char *)&server.sin_addr,
(char *)hp->h_addr, hp->h_length );
        portNum          = 19; /* 'chargen' */
        server.sin_port = htons(portNum);
        server_length   = sizeof(server);
        status = connect( sock, (struct sockaddr
*)&server, server_length );
        if ( status < 0 ) {
            if ( retry_cnt++ > 1 ) {
                mti_PrintFormatted( "Error connecting
to server %s:%d\n",
                                    hostname, portNum
);
                close( sock );
            } else {
                strcpy( hostname, "map" );  /* Put
your hostname here. */
            }
        }
    }

#ifdef WIN32
    {
        unsigned long non_blocking = 1;
        status = ioctlsocket( sock, FIONBIO,
&non_blocking );
        if ( status == SOCKET_ERROR ) {
            perror( "Setting socket status" );
        }
    }
#else
    statusFlags = fcntl( sock, F_GETFL );
    if ( statusFlags == SOCKET_ERROR ) {
        perror( "Getting socket status" );
    } else {
        int ctlValue;
        statusFlags |= O_NONBLOCK;
        ctlValue = fcntl( sock, F_SETFL, statusFlags
);
```

```
      if ( ctlValue == SOCKET_ERROR ) {
        perror( "Setting socket status" );
      }
    }
  }
#endif
  mti_AddLoadDoneCB( (mtiVoidFuncPtrT)loadDoneCB,
(void *)sock );
}
```

HDL code

```
entity top is
end top;

architecture a of top is
   signal s1 : bit := '0';
begin
   s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c -foreign "initForeign ./for_model.sl" top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign ./for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
# Load Done: Adding socket callback.
# Read returned 1 - Read
# Read returned 1 - Read !
# Read returned 1 - Read "
# Read returned 1 - Read #
# Read returned 1 - Read $
# Read returned 1 - Read %
# Read returned 1 - Read &
# Read returned 1 - Read '
# Read returned 1 - Read (
# Read returned 1 - Read )
# Read returned 1 - Read *
# Read returned 1 - Read +
# Read returned 1 - Read ,
# Read returned 1 - Read -
# Read returned 1 - Read .
# Read returned 1 - Read /
# Read returned 1 - Read 0
# Read returned 1 - Read 1
# Read returned 1 - Read 2
# Read returned 1 - Read 3
# Read returned 1 - Read 4
# Read returned 1 - Read 5
# Read returned 1 - Read 6
# Read returned 1 - Read 7
# Read returned 1 - Read 8
# Read returned 1 - Read 9
# Read returned 1 - Read :
# Read returned 1 - Read ;
# Read returned 1 - Read <
# Read returned 1 - Read =
# Read returned 1 - Read >
# Read returned 1 - Read ?
# Read returned 1 - Read @
# Read returned 1 - Read A
# Read returned 1 - Read B
# Read returned 1 - Read C
# Closing socket
VSIM 1> quit
```

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddSocketOutputReadyCB()

Adds or removes a socket output ready callback.

Syntax

mti_AddSocketOutputReadyCB(socket_desc, func, param)

Arguments

Name	Type	Description
socket_desc	int	A socket descriptor
func	mtiVoidFuncPtrT	A pointer to a function to be called whenever the socket descriptor is available for writing
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddSocketOutputReadyCB() puts a watch on the specified socket descriptor. Whenever the socket descriptor is available for writing, the simulator calls the specified function along with its parameter.

To remove a previously added callback, call mti_AddSocketOutputReadyCB() with the same socket descriptor but with a NULL function pointer.

mti_AddSocketInputReadyCB() and mti_AddSocketOutputReadyCB() are useful in setting up cosimulation environments where FLI code uses sockets to communicate with other processes. In the course of initialization, a cosimulation application typically would use standard system library routines to create or open a socket and obtain a socket descriptor and then call mti_AddSocketInputReadyCB() and mti_AddSocketOutputReadyCB() to set up the callback functions. During simulation, FLI code may initiate a non-blocking I/O operation on the socket (again using standard system library routines) and immediately return control to the simulator. When the I/O is completed, the simulator invokes the callback function which could check for errors, handle received data, or initiate another non-blocking I/O operation before returning to the simulator.

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_AddTclCommand()

Adds a user-defined, Tcl-style simulator command.

Syntax

mti_AddTclCommand(cmd_name, cmd_func, cmd_param, func_delete_cb)

Arguments

Name	Type	Description
cmd_name	char *	The name of the command being added
cmd_func	Tcl_CmdProc *	A pointer to a function that will be called whenever the command is recognized by the command interpreter
cmd_param	void *	A parameter to be passed to the command function; OPTIONAL - can be NULL
func_delete_cb	mtiVoidFuncPtrT	A pointer to a function that will be called if the command is redefined or deleted; OPTIONAL - can be NULL

Return Values

Nothing

Description

mti_AddTclCommand() adds the specified Tcl command to the simulator. The case of the command name is significant. The simulator command interpreter subsequently recognizes the command and calls the command function along with its parameter and user-supplied arguments whenever the command is recognized. The command function must return a valid Tcl status (for example TCL_OK or TCL_ERROR). The command function prototype is:

```
int commandFuncName( ClientData cmd_param, Tcl_Interp * interp, int argc,
char ** argv)
```

You can add a command with the same name as a previously added command (or even a standard simulator command), but only the last command added has any effect.

If you give the mti_AddTclCommand() function a non-NULL delete function callback (the func_delete_cb parameter), then the simulator calls this callback function when the command is deleted or redefined. If a restart occurs, or if another design is loaded, then the simulator deletes the command prior to the restart or design load, and this will also cause a call to func_delete_cb.

The delete callback function prototype is:

```
void deleteCBname( ClientData cmd_param )
```

To make the prototype of mti_AddTclCommand() visible, you must include the header file *tcl.h* in the FLI application code before *mti.h*.

Examples

FLI code

```
#include <stdlib.h>
#include <tcl.h>
#include <mti.h>
typedef struct {
  char         model_name[100];
  mtiSignalIdT sig1;
  mtiSignalIdT sig2;
} instanceInfoT;
int noAction( ClientData param, Tcl_Interp
* interp, int argc, char ** argv )
{
  mti_PrintFormatted( "Time [%ld,%ld] delta
%d:\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
  mti_PrintMessage( "  The printSigs command
has been deactivated.\n" );
  return TCL_OK;
}
void printSigInfo( mtiSignalIdT sigid, char
printFullName )
{
  char * region_name;
  mti_PrintFormatted( "   Signal " );
  if ( printFullName ) {
    region_name = mti_GetRegionFullName(
mti_GetSignalRegion( sigid ));
    mti_PrintFormatted( "%s/", region_name
);
    mti_VsimFree( region_name );
  }
  mti_PrintFormatted( "%s = ", mti_GetSignalName(
sigid ) );
  switch ( mti_GetTypeKind( mti_GetSignalType(
sigid )) ) {
    case MTI_TYPE_SCALAR:
    case MTI_TYPE_ENUM:
    case MTI_TYPE_PHYSICAL:
      mti_PrintFormatted( "%d\n", mti_GetSignalValue(
sigid ) );
      break;
    default:
      mti_PrintFormatted( "(Type not supported)\n"
);
      break;
  }
}
int printRegionInfo( ClientData param, Tcl_Interp
* interp,
                     int argc, char ** argv
)
{
  instanceInfoT * inst_info = (instanceInfoT*)param;
  char           printFullName = 0;
```

```
  if ( argc > 1 ) {
    if ( strcmp( argv[1], "full" ) == 0 )
{
        printFullName = 1;
    } else {
        Tcl_SetResult( interp, "printRegionInfo():
Unknown argument",
                       TCL_STATIC );
        return TCL_ERROR;
    }
  }
  mti_PrintFormatted( "Time [%ld,%ld] delta
%d:\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
  mti_PrintFormatted( "  Signal info for
%s:\n", inst_info->model_name );
  printSigInfo( inst_info->sig1, printFullName
);
  printSigInfo( inst_info->sig2, printFullName
);
  if ( mti_Now() > 15 ) {
      mti_AddTclCommand( "printSigs", noAction,
0, 0 );
  }
  return TCL_OK;
}
void deleteCB( ClientData param )
{
  instanceInfoT * inst_info = (instanceInfoT*)param;
  mti_PrintFormatted( "Time [%ld,%ld] delta
%d:\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
  mti_PrintFormatted( "  Deleting old command
data for %s.\n",
                     inst_info->model_name
);
}
void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}
void initForeign(
  mtiRegionIdT      region,    /* The ID
of the region in which this     */
                               /* foreign
architecture is instantiated.  */
  char            *param,    /* The last
part of the string in the     */
                               /* foreign
attribute.                    */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.  */
)
```

```
{
  instanceInfoT     * inst_info;
  mtiInterfaceListT * portp;
  inst_info = (instanceInfoT *)malloc( sizeof(instanceInfoT)
);
  /* ASSUME param is less than 100 chars
and
   * there are at least two signal ports.
   */
  strcpy( inst_info->model_name, param );
  portp = ports;
  inst_info->sig1 = portp->u.port;
  portp = portp->nxt;
  inst_info->sig2 = portp->u.port;
mti_AddTclCommand( "printSigs", printRegionInfo,
inst_info, deleteCB );
mti_AddQuitCB( cleanupCallback, inst_info
);
mti_AddRestartCB( cleanupCallback, inst_info
);
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity for_model is
  port ( inb : in bit;
         ins : in std_logic
       );
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : std_logic := '1';

  component for_model is
    port ( inb : in bit;
           ins : in std_logic
         );
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model
      port map ( s1, s2 );

  s1 <= not s1 after 5 ns;
  s2 <= not s2 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.5 Dev

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> printSigs
# Time [0,0] delta 0:
#   Signal info for for_model:
#     Signal s1 = 0
#     Signal s2 = 3
VSIM 2> printSigs full
# Time [0,0] delta 0:
#   Signal info for for_model:
#     Signal /top/s1 = 0
#     Signal /top/s2 = 3
VSIM 3> run 5
VSIM 4> printSigs
# Time [0,5] delta 1:
#   Signal info for for_model:
#     Signal s1 = 1
#     Signal s2 = 2
VSIM 5> printSigs all
# printRegionInfo(): Unknown argument
VSIM 6> run 5
VSIM 7> printSigs
# Time [0,10] delta 1:
#   Signal info for for_model:
#     Signal s1 = 0
#     Signal s2 = 3
VSIM 8> run 10
VSIM 9> printSigs
# Time [0,20] delta 1:
#   Signal info for for_model:
#     Signal s1 = 0
#     Signal s2 = 3
# Time [0,20] delta 1:
#   Deleting old command data for for_model.
VSIM 10> run 5
VSIM 11> printSigs
# Time [0,25] delta 1:
#   The printSigs command has been deactivated.
VSIM 12> quit
# Cleaning up...
```

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mti_AddUCDBSaveCB()

Adds the specified function to the coverage save callback list of the simulator.

Syntax

mti_AddUCDBSaveCB (region, func, param)

Arguments

Name	Type	Description
region	mtiRegionIdT	Valid handle for the region on which the function will be called.
func	mtiUCDBSaveFuncPtrT	Function pointer which will be called on the specified region.
param	void *	The parameter to the function. Can also be null.

Return Values

Nothing

Description

The Coverage Save command will call this function on the specified region.

mti_AskStdin()

Prompts the user for an input string.

Syntax

error_code = mti_AskStdin(buffer, prompt)

Arguments

Name	Type	Description
buffer	char *	A pointer to a character buffer in which the user's input is returned
prompt	char *	A character string that will be used as the prompt to the user

Return Values

Name	Type	Description
error_code	int	-1 if the buffer parameter is NULL; 0 otherwise

Description

mti_AskStdin() gets input from the user by displaying the specified prompt on the vsim command line and returning what the user types. All characters entered up to, but not including, a newline character are returned in the character buffer. The character string is null-terminated. The caller is responsible for allocating the space for the buffer.

Examples

FLI code

```
#include <strings.h>#include <mti.h>
void printSigInfo( void * param )
{
  char        buffer[128];
  int         done = 0;
  mtiSignalIdT sigid;
  while ( ! done ) {
    mti_AskStdin( buffer, "printSigs:" );
    if ( strcasecmp( buffer, "quit" ) ==
0 ) {
        done = 1;
    } else {
        sigid = mti_FindSignal( buffer );
        if ( ! sigid ) {
            mti_PrintFormatted( "    Signal
%s not found.\n", buffer );
        } else {
            switch ( mti_GetTypeKind( mti_GetSignalType(
sigid )) ) {
                case MTI_TYPE_SCALAR:
                case MTI_TYPE_ENUM:
                case MTI_TYPE_PHYSICAL:
                  mti_PrintFormatted( "
  Signal %s = %d\n", buffer,
                                        mti_GetSignalValue(
sigid ) );
                  break;
                default:
                  mti_PrintFormatted( "
  The type of signal %s "
                                        "is
not supported.\n", buffer );
                  break;
            }
        }
    }
  }
}
void initForeign(
  mtiRegionIdT      region,   /* The ID
of the region in which this      */
                                  /* foreign
architecture is instantiated.  */
  char              *param,   /* The last
part of the string in the      */
                                  /* foreign
attribute.                      */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.   */
)
{
  mti_AddCommand( "printSigs", printSigInfo
```

```
);
}
```

HDL code

```
entity top is
end top;

architecture a of top is
  signal s1 : bit := '0';
  signal s2 : real := 1.0;
begin
  s1 <= not s1 after 5 ns;
  s2 <= s2 + 1.0 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> printSigs
printSigs: /top/s1
/top/s1
#    Signal /top/s1 = 0
printSigs: /top/s2
/top/s2
#    The type of signal /top/s2 is not supported.
printSigs: /top/s3
/top/s3
#    Signal /top/s3 not found.
printSigs: quit
quit
VSIM 2> run 5
VSIM 3> quit
```

mti_Break()

Requests the simulator to halt.

Syntax

mti_Break()

Arguments

None

Return Values

Nothing

Description

mti_Break() requests the simulator to halt the simulation and issue an assertion message with the text "Simulation halt requested by foreign interface". The break request is satisfied after the foreign code returns control to the simulator. You can continue the simulation after it has been halted with mti_Break().

You cannot call mti_Break() during elaboration.

Examples

FLI code

```
#include <mti.h>
typedef enum {
  STD_LOGIC_U,        /* 'U' */
  STD_LOGIC_X,        /* 'X' */
  STD_LOGIC_0,        /* '0' */
  STD_LOGIC_1,        /* '1' */
  STD_LOGIC_Z,        /* 'Z' */
  STD_LOGIC_W,        /* 'W' */
  STD_LOGIC_L,        /* 'L' */
  STD_LOGIC_H,        /* 'H' */
  STD_LOGIC_D         /* '-' */
} StdLogicT;
void monitorSignal( void * param )
{
  mtiSignalIdT sigid = (mtiSignalIdT)param;
  switch ( mti_GetSignalValue( sigid ) )
{
    case STD_LOGIC_X:
    case STD_LOGIC_W:
      mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s is UNKNOWN\n",
                          mti_NowUpper(),
mti_Now(), mti_Delta(),
                          mti_GetSignalName(
sigid ) );
      mti_Break();
      break;
    default:
      break;
  }
}
void initForeign(
  mtiRegionIdT       region,   /* The ID
of the region in which this      */
                                /* foreign
architecture is instantiated.  */
  char              *param,    /* The last
part of the string in the      */
                                /* foreign
attribute.                     */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.   */
)
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;
  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "SignalMonitor",
monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT
);
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is
   signal s1 : std_logic := '0';
begin
  p1 : process
  begin
    c1 : case s1 is
       when 'U' => s1 <= 'X' after 5 ns;
       when 'X' => s1 <= '0' after 5 ns;
       when '0' => s1 <= '1' after 5 ns;
       when '1' => s1 <= 'Z' after 5 ns;
       when 'Z' => s1 <= 'W' after 5 ns;
       when 'W' => s1 <= 'L' after 5 ns;
       when 'L' => s1 <= 'H' after 5 ns;
       when 'H' => s1 <= '-' after 5 ns;
       when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    wait for 5 ns;
  end process;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,15] delta 0: Signal s1 is UNKNOWN
# Simulation halt requested by foreign interface
# Stopped at top.vhd line 27
VSIM 2> drivers /top/s1
# Drivers for /top/s1:
#  W : Signal /top/s1
#    W : Driver /top/p1
#
VSIM 3> cont
# Time [0,40] delta 0: Signal s1 is UNKNOWN
# Simulation halt requested by foreign interface
# Stopped at top.vhd line 27
VSIM 4> drivers /top/s1
# Drivers for /top/s1:
#  X : Signal /top/s1
#    X : Driver /top/p1
#
VSIM 5> quit
```

Related Topics

mti_Interp()

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mti_Cmd()

Executes a simulator command with Tcl return status and no transcribing.

Syntax

tcl_status = mti_Cmd(command)

Arguments

Name	Type	Description
command	char *	A simulator command

Return Values

Name	Type	Description
tcl_status	int	TCL_OK if the command is successful or TCL_ERROR if there is an error

Description

mti_Cmd() instructs the simulator to execute the specified command. The string must contain the command just as it would be typed at the VSIM prompt. The simulator does not transcribe the results of the command into the vsim transcript, but you can obtain them by using the Tcl_interp pointer. (See mti_Interp()). You should reset the command result using Tcl_ResetResult() after each call to mti_Cmd().

You cannot send any command that changes the state of simulation (such as run, restart, restore, and so on) from a foreign architecture, foreign subprogram, or callback that is executing under the direct control of vsim.

Examples

FLI code

```c
#include <stdio.h>
#include <stdlib.h>
#include <tcl.h>
#include <mti.h>
typedef enum {
  STD_LOGIC_U,        /* 'U' */
  STD_LOGIC_X,        /* 'X' */
  STD_LOGIC_0,        /* '0' */
  STD_LOGIC_1,        /* '1' */
  STD_LOGIC_Z,        /* 'Z' */
  STD_LOGIC_W,        /* 'W' */
  STD_LOGIC_L,        /* 'L' */
  STD_LOGIC_H,        /* 'H' */
  STD_LOGIC_D         /* '-' */
} StdLogicT;
void monitorSignal( void * param )
{
  char          buffer[256];
  char *        region_name;
  char *        signal_name;
  int           status;
  mtiSignalIdT sigid = (mtiSignalIdT)param;
  Tcl_Interp * interp;
  switch ( mti_GetSignalValue( sigid ) )
{
    case STD_LOGIC_X:
    case STD_LOGIC_W:
      signal_name = mti_GetSignalName( sigid
);
      region_name = mti_GetRegionFullName(
mti_GetSignalRegion( sigid ));
      mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is UNKNOWN\n",
                          mti_NowUpper(), mti_Now(),
mti_Delta(),
                          region_name, signal_name
);
      sprintf( buffer, "drivers %s/%s", region_name,
signal_name );
      interp = mti_Interp();
      status = mti_Cmd( buffer );
      if ( status != TCL_OK ) {
        mti_PrintMessage( "ERROR while executing
drivers command.\n" );
      } else {
        mti_PrintFormatted( "The drivers
of %s/%s are:\n%s\n",
                region_name, signal_name,
Tcl_GetStringResult(interp) );
      }
      Tcl_ResetResult( interp );
      mti_VsimFree( region_name );
      break;
    default:
```

```
        break;
    }
}
void initForeign(
  mtiRegionIdT        region,    /* The ID
of the region in which this      */
                                 /* foreign
architecture is instantiated.  */
  char              *param,    /* The last
part of the string in the      */
                                 /* foreign
attribute.                      */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports      /* A list
of ports for the foreign model.   */
)
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;
  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "SignalMonitor",
monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT
);
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is
  signal s1 : std_logic := '0';
begin
  p1 : process
  begin
    c1 : case s1 is
      when 'U' => s1 <= 'X' after 5 ns;
      when 'X' => s1 <= '0' after 5 ns;
      when '0' => s1 <= '1' after 5 ns;
      when '1' => s1 <= 'Z' after 5 ns;
      when 'Z' => s1 <= 'W' after 5 ns;
      when 'W' => s1 <= 'L' after 5 ns;
      when 'L' => s1 <= 'H' after 5 ns;
      when 'H' => s1 <= '-' after 5 ns;
      when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    wait for 5 ns;
  end process;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,15] delta 0: Signal /top/s1 is UNKNOWN
# The drivers of /top/s1 are:
# Drivers for /top/s1:
#  W : Signal /top/s1
#    W : Driver /top/p1
#
# Time [0,40] delta 0: Signal /top/s1 is UNKNOWN
# The drivers of /top/s1 are:
# Drivers for /top/s1:
#  X : Signal /top/s1
#    X : Driver /top/p1
#
VSIM 2> quit
```

Related Topics

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CallStack()

Prints a call stack from the point where the call is made.

Syntax

int mti_CallStack()

Arguments

Return Values

- A non-zero value if the call is successful.

- A zero value (0) if the call failed.

Description

mti_CallStack() produces a call stack from the point where the call is made and the output appears in the vsim transcript window. The call stack covers both user C code and Verilog code.

mti_Command()

Executes a simulator command.

Syntax

mti_Command(command)

Arguments

Name	Type	Description
command	char *	A simulator command

Return Values

Nothing

Description

mti_Command() instructs the simulator to execute the specified command. The string must contain the command just as it would be typed at the VSIM prompt. The simulator transcribes the results of the command in the vsim transcript.

You cannot send any command that changes the state of simulation (such as run, restart, restore, and so on) from a foreign architecture, foreign subprogram, or callback that is executing under the direct control of vsim.

Examples

FLI code

```c
#include <stdio.h>
#include <stdlib.h>
#include <mti.h>
typedef enum {
  STD_LOGIC_U,        /* 'U' */
  STD_LOGIC_X,        /* 'X' */
  STD_LOGIC_0,        /* '0' */
  STD_LOGIC_1,        /* '1' */
  STD_LOGIC_Z,        /* 'Z' */
  STD_LOGIC_W,        /* 'W' */
  STD_LOGIC_L,        /* 'L' */
  STD_LOGIC_H,        /* 'H' */
  STD_LOGIC_D         /* '-' */
} StdLogicT;
void monitorSignal( void * param )
{
  char          buffer[256];
  char *        region_name;
  char *        signal_name;
  mtiSignalIdT sigid = (mtiSignalIdT)param;
  switch ( mti_GetSignalValue( sigid ) )
{
    case STD_LOGIC_X:
    case STD_LOGIC_W:
      signal_name = mti_GetSignalName( sigid
);
      region_name = mti_GetRegionFullName(
mti_GetSignalRegion( sigid ));
      mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is UNKNOWN\n",
                          mti_NowUpper(),
mti_Now(), mti_Delta(),
                          region_name, signal_name
);
      sprintf( buffer, "drivers %s/%s", region_name,
signal_name );
      mti_Command( buffer );
      mti_VsimFree( region_name );
      break;
    default:
      break;
  }
}
void initForeign(
  mtiRegionIdT      region,   /* The ID
of the region in which this      */
                                    /* foreign
architecture is instantiated.  */
  char              *param,   /* The last
part of the string in the      */
                                    /* foreign
attribute.                      */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
```

```
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.   */
)
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;
  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "SignalMonitor",
monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT
);
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is
  signal s1 : std_logic := '0';
begin
  p1 : process
  begin
    c1 : case s1 is
      when 'U' => s1 <= 'X' after 5 ns;
      when 'X' => s1 <= '0' after 5 ns;
      when '0' => s1 <= '1' after 5 ns;
      when '1' => s1 <= 'Z' after 5 ns;
      when 'Z' => s1 <= 'W' after 5 ns;
      when 'W' => s1 <= 'L' after 5 ns;
      when 'L' => s1 <= 'H' after 5 ns;
      when 'H' => s1 <= '-' after 5 ns;
      when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    wait for 5 ns;
  end process;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,15] delta 0: Signal /top/s1 is UNKNOWN
# Drivers for /top/s1:
#  W : Signal /top/s1
#    W : Driver /top/p1
#
# Time [0,40] delta 0: Signal /top/s1 is UNKNOWN
# Drivers for /top/s1:
#  X : Signal /top/s1
#    X : Driver /top/p1
#
VSIM 2> quit
```

Related Topics

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateArrayType()

Creates an array type.

Syntax

type_id = mti_CreateArrayType(left, right, element_type)

Arguments

Name	Type	Description
left	mtiInt32T	The left bound of the new array type
right	mtiInt32T	The right bound of the new array type
element_type	mtiTypeIdT	The type of the elements of the new array type

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the new array type

Description

mti_CreateArrayType() creates a new type ID that describes a VHDL array type whose bounds are the specified left and right values and whose elements are of the specified element type.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>
typedef enum {
  SIGVAL_0,
  SIGVAL_1,
  SIGVAL_X
} mySigType;
char *enum_lits[3] = { "0", "1", "X" };
typedef struct {
  mtiSignalIdT sigid1;
  mtiSignalIdT sigid2;
  mtiSignalIdT sigid3;
  mtiDriverIdT drvid1;
  mtiDriverIdT drvid2;
  mtiDriverIdT drvid3;
} instanceInfoT;
/* This function inverts mySig(2) every 5
ns. */
void driveSignal1( void * param )
{
  char          * region_name;
  char          * signal_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;
  sigval = mti_GetSignalValue( inst->sigid1
);
  switch ( sigval ) {
    case SIGVAL_0:
      mti_ScheduleDriver( inst->drvid1, SIGVAL_1,
5, MTI_INERTIAL );
    break;
    case SIGVAL_1:
      mti_ScheduleDriver( inst->drvid1, SIGVAL_0,
5, MTI_INERTIAL );
      break;
    case SIGVAL_X:
      signal_name = mti_GetSignalNameIndirect(
inst->sigid1, NULL, 0 );
      region_name =
        mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
      mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is UNKNOWN\n",
                          mti_NowUpper(),
mti_Now(), mti_Delta(),
                          region_name, signal_name
);
      mti_VsimFree( signal_name );
      mti_VsimFree( region_name );
      break;
    default:
      signal_name = mti_GetSignalNameIndirect(
inst->sigid1, NULL, 0 );
      region_name =
        mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
```

```
      mti_PrintFormatted( "Time [%d,%d] delta
%d: "
                          "Unexpected value
%d on signal %s/%s\n",
                          mti_NowUpper(),
mti_Now(), mti_Delta(),
                          sigval, region_name,
signal_name );
      mti_VsimFree( signal_name );
      mti_VsimFree( region_name );
      break;
  }
}
/* This function inverts mySig(1) every 10
ns. */
void driveSignal2( void * param )
{
  char          * region_name;
  char          * signal_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;
  sigval = mti_GetSignalValue( inst->sigid2
);
  switch ( sigval ) {
    case SIGVAL_0:
      mti_ScheduleDriver( inst->drvid2, SIGVAL_1,
10, MTI_INERTIAL );
    break;
    case SIGVAL_1:
      mti_ScheduleDriver( inst->drvid2, SIGVAL_0,
10, MTI_INERTIAL );
      break;
    case SIGVAL_X:
      signal_name = mti_GetSignalNameIndirect(
inst->sigid2, NULL, 0 );
      region_name =
        mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
      mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is UNKNOWN\n",
                          mti_NowUpper(),
mti_Now(), mti_Delta(),
                          region_name, signal_name
);
      mti_VsimFree( signal_name );
      mti_VsimFree( region_name );
      break;
    default:
      signal_name = mti_GetSignalNameIndirect(
inst->sigid2, NULL, 0 );
      region_name =
        mti_GetRegionFullName(mti_GetSignalRegion
(inst->sigid1));
      mti_PrintFormatted( "Time [%d,%d] delta
%d: "
                          "Unexpected value
%d on signal %s/%s\n",
                          mti_NowUpper(),
mti_Now(), mti_Delta(),
```

```
                              sigval, region_name,
signal_name );
      mti_VsimFree( signal_name );
      mti_VsimFree( region_name );
      break;
   }
}
/* This function drives mySig(0) with the
values of mySig(2) and mySig(1). */
void driveSignal3( void * param )
{
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval1;
  mtiInt32T       sigval2;
  sigval1 = mti_GetSignalValue( inst->sigid1
);
  sigval2 = mti_GetSignalValue( inst->sigid2
);
  if ( sigval1 == sigval2 ) {
    mti_ScheduleDriver( inst->drvid3, sigval1,
0, MTI_INERTIAL );
  } else {
    mti_ScheduleDriver( inst->drvid3, SIGVAL_X,
0, MTI_INERTIAL );
  }
}
void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}
void initForeign(
  mtiRegionIdT       region,    /* The ID
of the region in which this      */
                                /* foreign
architecture is instantiated.  */
  char              *param,    /* The last
part of the string in the      */
                                /* foreign
attribute.                     */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.   */
)
{
  instanceInfoT * inst;
  mtiProcessIdT   procid;
  mtiSignalIdT  * elem_list;
  mtiSignalIdT    sigid;
  mtiTypeIdT      array_type;
  mtiTypeIdT      enum_type;
  inst       = (instanceInfoT *)malloc(
sizeof(instanceInfoT) );
  enum_type   = mti_CreateEnumType( 1, 3,
enum_lits );
  array_type  = mti_CreateArrayType( 2,
0, enum_type );
```

```
  sigid        = mti_CreateSignal( "mySig",
region, array_type );
  elem_list    = mti_GetSignalSubelements(
sigid, NULL );
  inst->sigid1 = elem_list[0];
  inst->drvid1 = mti_CreateDriver( inst->sigid1
);
  procid       = mti_CreateProcess( "mySig1Driver",
driveSignal1, inst );
  mti_Sensitize( procid, inst->sigid1, MTI_EVENT
);
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid1, procid
);
  inst->sigid2 = elem_list[1];
  inst->drvid2 = mti_CreateDriver( inst->sigid2
);
  procid       = mti_CreateProcess( "mySig2Driver",
driveSignal2, inst );
  mti_Sensitize( procid, inst->sigid2, MTI_EVENT
);
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid2, procid
);
  inst->sigid3 = elem_list[2];
  inst->drvid3 = mti_CreateDriver( inst->sigid3
);
  procid       = mti_CreateProcess( "mySig3Driver",
driveSignal3, inst );
  mti_Sensitize( procid, inst->sigid1, MTI_EVENT
);
  mti_Sensitize( procid, inst->sigid2, MTI_EVENT
);
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid3, procid
);
  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst
);
  mti_VsimFree( elem_list );
}
```

HDL code

```
entity top is
end top;

architecture a of top is
  signal s1 : bit := '0';
begin
  s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> examine mysig
# {0 0 0}
VSIM 2> run 5
VSIM 3> examine mysig
# {1 0 X}
VSIM 4> run 5
VSIM 5> examine mysig
# {0 1 X}
VSIM 6> run 5
VSIM 7> examine mysig
# {1 1 1}
VSIM 8> quit
# Cleaning up...
```

Related Topics

mti_SetSignalValue()

mti_SetDriverOwner()

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateDriver()

Creates a driver on a VHDL signal.

Syntax

driver_id = mti_CreateDriver(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL signal

Return Values

Name	Type	Description
driver_id	mtiDriverIdT	A handle to the new driver or NULL if there is an error

Description

mti_CreateDriver() creates a new driver for the specified array or scalar signal. You must create a driver for a resolved signal in order to be able to drive values onto that signal and have the values be resolved. You can create multiple drivers for a resolved signal, but no more than one driver can be created for an unresolved signal. Alternatively, you can change the values of an unresolved signal using mti_SetSignalValue() if that signal does not have any drivers.

When using mti_CreateDriver() it is necessary to follow up with a call to mti_SetDriverOwner(); otherwise, the vsim drivers command and the Dataflow window may give unexpected or incorrect information regarding the newly created driver.

You cannot create a driver cannot on a signal of type record, but you can create drivers on non-record subelements of a record signal.

You cannot create a driver on a subelement of a resolved composite signal. You must create drivers at the resolution level or above.

mti_CreateDriver() cannot create a driver on a VHDL port that has not been collapsed with the connected signal. A VHDL port is not collapsed when it is connected to a Verilog signal, when a conversion function appears in a VHDL port map, or when the vsim option -nocollapse is used.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>
typedef enum {
    STD_LOGIC_U,
    STD_LOGIC_X,
    STD_LOGIC_0,
    STD_LOGIC_1,
    STD_LOGIC_Z,
    STD_LOGIC_W,
    STD_LOGIC_L,
    STD_LOGIC_H,
    STD_LOGIC_D
} standardLogicType;
typedef struct {
    mtiSignalIdT sigid;
    mtiDriverIdT drvid;
    mtiTypeIdT   time_type;
} instanceInfoT;
void driveScalarSignal( void * param )
{
    char         * curr_time_str;
    char         * region_name;
    instanceInfoT * inst = (instanceInfoT*)param;
    mtiInt32T       sigval;
    mtiTime64T      curr_time;
    region_name   = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid));
    sigval        = mti_GetSignalValue( inst->sigid
);
    curr_time_str = mti_Image( mti_NowIndirect(&curr_time),
inst->time_type );
    mti_PrintFormatted( "Time %s delta %d:
Signal %s/%s is %s\n",
                        curr_time_str, mti_Delta(),
                        region_name, mti_GetSignalName(
inst->sigid ),
                        mti_SignalImage(inst->sigid)
);
    switch ( sigval ) {
      case STD_LOGIC_U:  sigval = STD_LOGIC_X;
 break;
      case STD_LOGIC_X:  sigval = STD_LOGIC_0;
 break;
      case STD_LOGIC_0:  sigval = STD_LOGIC_1;
 break;
      case STD_LOGIC_1:  sigval = STD_LOGIC_Z;
 break;
      case STD_LOGIC_Z:  sigval = STD_LOGIC_W;
 break;
      case STD_LOGIC_W:  sigval = STD_LOGIC_L;
 break;
      case STD_LOGIC_L:  sigval = STD_LOGIC_H;
 break;
      case STD_LOGIC_H:  sigval = STD_LOGIC_D;
 break;
```

```
        case STD_LOGIC_D:  sigval = STD_LOGIC_U;
 break;
        default:           sigval = STD_LOGIC_U;
 break;
    }
    mti_ScheduleDriver( inst->drvid, sigval,
5, MTI_INERTIAL );
    mti_VsimFree( region_name );
}
void driveArraySignal( void * param )
{
    char         * curr_time_str;
    char         * region_name;
    char         * sigval;
    instanceInfoT * inst = (instanceInfoT*)param;
    int            i;
    mtiTime64T     curr_time;
    region_name  = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid));
    sigval       = (char *)mti_GetArraySignalValue(
inst->sigid, 0 );
    curr_time_str = mti_Image( mti_NowIndirect(&curr_time),
inst->time_type );
    mti_PrintFormatted( "Time %s delta %d:
Signal %s/%s is %s\n",
                        curr_time_str, mti_Delta(),
                        region_name, mti_GetSignalName(
inst->sigid ),
                        mti_SignalImage(inst->sigid)
);
    for ( i = 0; i < mti_TickLength( mti_GetSignalType(
inst->sigid )); i++ ) {
        switch ( sigval[i] ) {
          case STD_LOGIC_U:  sigval[i] =
STD_LOGIC_X;  break;
          case STD_LOGIC_X:  sigval[i] =
STD_LOGIC_0;  break;
          case STD_LOGIC_0:  sigval[i] =
STD_LOGIC_1;  break;
          case STD_LOGIC_1:  sigval[i] =
STD_LOGIC_Z;  break;
          case STD_LOGIC_Z:  sigval[i] =
STD_LOGIC_W;  break;
          case STD_LOGIC_W:  sigval[i] =
STD_LOGIC_L;  break;
          case STD_LOGIC_L:  sigval[i] =
STD_LOGIC_H;  break;
          case STD_LOGIC_H:  sigval[i] =
STD_LOGIC_D;  break;
          case STD_LOGIC_D:  sigval[i] =
STD_LOGIC_U;  break;
          default:           sigval[i] =
STD_LOGIC_U;  break;
        }
    }
    mti_ScheduleDriver( inst->drvid, (long)sigval,
5, MTI_INERTIAL );
    mti_VsimFree( sigval );
    mti_VsimFree( region_name );
```

```
    }
void initForeign(
    mtiRegionIdT        region,    /* The ID
of the region in which this        */
                                   /* foreign
architecture is instantiated.     */
    char              *param,     /* The last
part of the string in the        */
                                   /* foreign
attribute.                        */
    mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
    mtiInterfaceListT *ports      /* A list
of ports for the foreign model.   */
)
{
    instanceInfoT * inst;
    mtiProcessIdT   procid;
    inst       = (instanceInfoT *)mti_Malloc(
sizeof(instanceInfoT) );
    inst->sigid = mti_FindSignal( "/top/s1"
);
    inst->drvid = mti_CreateDriver( inst->sigid
);
    procid      = mti_CreateProcess( "sigDriver1",
driveScalarSignal, inst );
    mti_Sensitize( procid, inst->sigid, MTI_EVENT
);
    mti_ScheduleWakeup( procid, 0 );
    mti_SetDriverOwner( inst->drvid, procid
);
    inst->time_type = mti_CreateTimeType();
    inst       = (instanceInfoT *)mti_Malloc(
sizeof(instanceInfoT) );
    inst->sigid = mti_FindSignal( "/top/s2"
);
    inst->drvid = mti_CreateDriver( inst->sigid
);
    procid      = mti_CreateProcess( "sigDriver2",
driveArraySignal, inst );
    mti_Sensitize( procid, inst->sigid, MTI_EVENT
);
    mti_ScheduleWakeup( procid, 0 );
    mti_SetDriverOwner( inst->drvid, procid
);
    inst->tie_type = mti_CreateTimeType();
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';
  signal s2 : std_logic_vector( 3 downto 0 ) := "UX01";

begin
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 42
# Time {0 ns} delta 1: Signal /top/s1 is '0'
# Time {0 ns} delta 1: Signal /top/s2 is "UX01"
# Time {5 ns} delta 0: Signal /top/s2 is "X01Z"
# Time {5 ns} delta 0: Signal /top/s1 is '1'
# Time {10 ns} delta 0: Signal /top/s1 is 'Z'
# Time {10 ns} delta 0: Signal /top/s2 is "01ZW"
# Time {15 ns} delta 0: Signal /top/s2 is "1ZWL"
# Time {15 ns} delta 0: Signal /top/s1 is 'W'
# Time {20 ns} delta 0: Signal /top/s1 is 'L'
# Time {20 ns} delta 0: Signal /top/s2 is "ZWLH"
# Time {25 ns} delta 0: Signal /top/s2 is "WLH-"
# Time {25 ns} delta 0: Signal /top/s1 is 'H'
# Time {30 ns} delta 0: Signal /top/s1 is '-'
# Time {30 ns} delta 0: Signal /top/s2 is "LH-U"
# Time {35 ns} delta 0: Signal /top/s2 is "H-UX"
# Time {35 ns} delta 0: Signal /top/s1 is 'U'
# Time {40 ns} delta 0: Signal /top/s1 is 'X'
# Time {40 ns} delta 0: Signal /top/s2 is "-UX0"
VSIM 2> drivers /top/s1
# Drivers for /top/s1:
#    X  : Signal /top/s1
#      X : Driver /top/sigDriver1
#          0 at 45 ns
#
VSIM 3> drivers /top/s2
# Drivers for /top/s2(3:0):
#    -  : Signal /top/s2(3)
#      - : Driver /top/sigDriver2
#          U at 45 ns
#    U  : Signal /top/s2(2)
#      U : Driver /top/sigDriver2
#          X at 45 ns
#    X  : Signal /top/s2(1)
#      X : Driver /top/sigDriver2
#          0 at 45 ns
#    0  : Signal /top/s2(0)
#      0 : Driver /top/sigDriver2
#          1 at 45 ns
#
VSIM 4> quit
```

Related Topics

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateEnumType()

Creates an enumeration type.

Syntax

type_id = mti_CreateEnumType(size, count, literals)

Arguments

Name	Type	Description
size	mtiInt32T	The number of bytes to be used to store the values of the new enumeration type; if the count parameter is greater than 256 then size must be 4; otherwise size should be 1
count	mtiInt32T	The number of literals/values in the new enumeration type
literals	char **	An array of strings that define the enumeration literals for the new enumeration type

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the new enumeration type

Description

mti_CreateEnumType() creates a new type ID that describes a VHDL enumeration type. The new type consists of the specified enumeration literals and its values are of the specified size. The count parameter indicates the number of strings in the literals parameter. The left-most value of the enumeration type is 0 and is associated with the first literal string, the next value is 1 and is associated with the next literal string, and so on.

If there are more than 256 values in the enumeration type, then you must use 4 bytes to store the values; otherwise use 1 byte.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>
typedef enum {
  SIGVAL_0,
  SIGVAL_1,
  SIGVAL_X
} mySigType;
char *enum_lits[3] = { "0", "1", "X" };
typedef struct {
  mtiSignalIdT sigid1;
  mtiSignalIdT sigid2;
  mtiSignalIdT sigid3;
  mtiDriverIdT drvid1;
  mtiDriverIdT drvid2;
  mtiDriverIdT drvid3;
} instanceInfoT;
/* This function inverts mySig1 every 5 ns.
*/
void driveSignal1( void * param )
{
  char         * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;
  sigval = mti_GetSignalValue( inst->sigid1
);
  switch ( sigval ) {
   case SIGVAL_0:
    mti_ScheduleDriver( inst->drvid1, SIGVAL_1,
5, MTI_INERTIAL );
     break;
   case SIGVAL_1:
    mti_ScheduleDriver( inst->drvid1, SIGVAL_0,
5, MTI_INERTIAL );
     break;
   case SIGVAL_X:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
    mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is UNKNOWN\n",
                    mti_NowUpper(), mti_Now(),
mti_Delta(),
                    region_name, mti_GetSignalName(
inst->sigid1 ) );
   mti_VsimFree( region_name );
    break;
   default:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
    mti_PrintFormatted( "Time [%d,%d] delta
%d: "
                    "Unexpected value
%d on signal %s/%s\n",
                    mti_NowUpper(), mti_Now(),
mti_Delta(),
                    sigval, region_name,
                    mti_GetSignalName(
```

```
inst->sigid1 ) );
    mti_VsimFree( region_name );
    break;
  }
}
/* This function inverts mySig2 every 10
ns. */
void driveSignal2( void * param )
{
  char         * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T      sigval;
  sigval = mti_GetSignalValue( inst->sigid2
);
  switch ( sigval ) {
   case SIGVAL_0:
    mti_ScheduleDriver( inst->drvid2, SIGVAL_1,
10, MTI_INERTIAL );
    break;
   case SIGVAL_1:
    mti_ScheduleDriver( inst->drvid2, SIGVAL_0,
10, MTI_INERTIAL );
    break;
   case SIGVAL_X:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
    mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is UNKNOWN\n",
                       mti_NowUpper(), mti_Now(),
mti_Delta(),
                       region_name, mti_GetSignalName(
inst->sigid2 ) );
    mti_VsimFree( region_name );
    break;
   default:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
    mti_PrintFormatted( "Time [%d,%d] delta
%d: "
                       "Unexpected value
%d on signal %s/%s\n",
                       mti_NowUpper(), mti_Now(),
mti_Delta(),
                       sigval, region_name,
                       mti_GetSignalName(
inst->sigid2 ) );
    mti_VsimFree( region_name );
    break;
  }
}
/* This function drives mySig3 with the values
of mySig1 and mySig2. */
void driveSignal3( void * param )
{
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T      sigval1;
  mtiInt32T      sigval2;
  sigval1 = mti_GetSignalValue( inst->sigid1
);
   sigval2 = mti_GetSignalValue( inst->sigid2
```

```
  );
    if ( sigval1 == sigval2 ) {
      mti_ScheduleDriver( inst->drvid3, sigval1,
0, MTI_INERTIAL );
    } else {
      mti_ScheduleDriver( inst->drvid3, SIGVAL_X,
0, MTI_INERTIAL );
    }
}
void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}
void initForeign(
  mtiRegionIdT        region,    /* The ID
of the region in which this       */
                                 /* foreign
architecture is instantiated.  */
  char              *param,    /* The last
part of the string in the      */
                                 /* foreign
attribute.                       */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports      /* A list
of ports for the foreign model.   */
)
{
  instanceInfoT * inst;
  mtiProcessIdT   procid;
  mtiTypeIdT      enum_type;
  inst        = (instanceInfoT *)malloc(
sizeof(instanceInfoT) );
  enum_type     = mti_CreateEnumType( 1, 3,
enum_lits );
  inst->sigid1 = mti_CreateSignal( "mySig1",
region, enum_type );
  inst->drvid1 = mti_CreateDriver( inst->sigid1
);
  procid       = mti_CreateProcess( "mySig1Driver",
driveSignal1, inst );
  mti_Sensitize( procid, inst->sigid1, MTI_EVENT
);
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid1, procid
);
  inst->sigid2 = mti_CreateSignal( "mySig2",
region, enum_type );
  inst->drvid2 = mti_CreateDriver( inst->sigid2
);
  procid       = mti_CreateProcess( "mySig2Driver",
driveSignal2, inst );
  mti_Sensitize( procid, inst->sigid2, MTI_EVENT
);
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid2, procid
);
```

```
  inst->sigid3 = mti_CreateSignal( "mySig3",
region, enum_type );
  inst->drvid3 = mti_CreateDriver( inst->sigid3
);
  procid      = mti_CreateProcess( "mySig3Driver",
driveSignal3, inst );
  mti_Sensitize( procid, inst->sigid1, MTI_EVENT
);
  mti_Sensitize( procid, inst->sigid2, MTI_EVENT
);
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid3, procid
);
  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst
);
}
```

HDL code

```
entity top is
end top;

architecture a of top is
  signal s1 : bit := '0';
begin
  s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> examine mySig1 mySig2 mySig3
# 0 0 0
VSIM 2> run 5
VSIM 3> examine mySig1 mySig2 mySig3
# 1 0 X
VSIM 4> run 5
VSIM 5> examine mySig1 mySig2 mySig3
# 0 1 X
VSIM 6> run 5
VSIM 7> examine mySig1 mySig2 mySig3
# 1 1 1
VSIM 8> quit
# Cleaning up...
```

Related Topics

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateProcess()

Creates a new VHDL process.

Syntax

process_id = mti_CreateProcess(name, func, param)

Arguments

Name	Type	Description
name	char *	The name of the new VHDL process; OPTIONAL - can be NULL
func	mtiVoidFuncPtrT	A pointer to the function that will be executed as the body of the new process
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL

Return Values

Name	Type	Description
process_id	mtiProcessIdT	A handle to the new VHDL process or NULL if there is an error

Description

mti_CreateProcess() creates a new VHDL process with the specified name. If the name is non-NULL, then it appears in the Process window of the simulator; otherwise, it does not. The simulator calls the specified function along with its parameter whenever the process executes. The process executes either at the time specified in a call to mti_ScheduleWakeup() or whenever one of the signals to which it is sensitive changes (see mti_Sensitize()).

If you create the process during elaboration from inside of a foreign architecture instance, then the simulator aumtomatically execugtes the process once at time zero after initializing all signals. If you create the process either after elaboration is complete or from any other context (such as from an initialization function that executes as a result of the loading of a foreign shared library by the -foreign option to vsim), then the simulation does not run the process automatically but must be scheduled or sensitized.

mti_CreateProcess() allows you to create a process with an illegal HDL name. This is useful for integrators who provide shared libraries for use by end customers, as this is an easy way to avoid potential name conflicts with HDL processes. We recommend the following naming style:

```
<PREFIX_name>
```

where PREFIX is 3 or 4 characters that denote your software (to avoid name conflicts with other integration software) and *name* is the name of the process. Enclosing the entire name in angle brackets makes it an illegal HDL name. For example, <MTI_foreign_architecture>.

We strongly recommend that you do not use characters in the name that will cause Tcl parsing problems. This includes spaces, the path separator (normally '/' or '.'), square brackets ([]), and dollar signs ($). If you must use these characters, then create an escaped name by putting a backslash (\) at both ends of the name.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>
typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;
typedef struct {
  mtiSignalIdT sigid;
  mtiDriverIdT drvid;
} instanceInfoT;
char * convertStdLogicValue( mtiInt32T sigval
)
{
  char * retval;
  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}
void driveSignal( void * param )
{
  char          * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;
  region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid));
  sigval = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta
%d: Signal %s/%s is %s\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta(),
                      region_name, mti_GetSignalName(
inst->sigid ),
                      convertStdLogicValue(
sigval ) );
  switch ( sigval ) {
    case STD_LOGIC_U:  sigval = STD_LOGIC_X;  break;
    case STD_LOGIC_X:  sigval = STD_LOGIC_0;  break;
    case STD_LOGIC_0:  sigval = STD_LOGIC_1;  break;
```

```
      case STD_LOGIC_1:  sigval = STD_LOGIC_Z;  break;
      case STD_LOGIC_Z:  sigval = STD_LOGIC_W;  break;
      case STD_LOGIC_W:  sigval = STD_LOGIC_L;  break;
      case STD_LOGIC_L:  sigval = STD_LOGIC_H;  break;
      case STD_LOGIC_H:  sigval = STD_LOGIC_D;  break;
      case STD_LOGIC_D:  sigval = STD_LOGIC_U;  break;
      default:  sigval = STD_LOGIC_U;  break;
   }
   mti_ScheduleDriver( inst->drvid, sigval,
5, MTI_INERTIAL );
   mti_VsimFree( region_name );
}
void cleanupCallback( void * param )
{
   mti_PrintMessage( "Cleaning up...\n" );
   free( param );
}
void initForeign(
   mtiRegionIdT       region,   /* The ID
of the region in which this      */
                                 /* foreign
architecture is instantiated.  */
   char              *param,    /* The last
part of the string in the      */
                                 /* foreign
attribute.                       */
   mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
   mtiInterfaceListT *ports      /* A list
of ports for the foreign model.   */
)
{
   instanceInfoT * inst;
   mtiProcessIdT   procid;
   inst       = (instanceInfoT *)malloc(
sizeof(instanceInfoT) );
   inst->sigid = mti_FindSignal( "/top/s1"
);
   inst->drvid = mti_CreateDriver( inst->sigid
);
   procid     = mti_CreateProcess( "sigDriver",
driveSignal, inst );
   mti_Sensitize( procid, inst->sigid, MTI_EVENT
);
   mti_SetDriverOwner( inst->drvid, procid
);
   mti_AddQuitCB( cleanupCallback, inst );
   mti_AddRestartCB( cleanupCallback, inst
);
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

   signal s1 : std_logic := '0';

   component for_model is
   end component;

   for all : for_model use entity work.for_model(a);

begin

   i1 : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,0] delta 0: Signal /top/s1 is '0'
# Time [0,5] delta 0: Signal /top/s1 is '1'
# Time [0,10] delta 0: Signal /top/s1 is 'Z'
# Time [0,15] delta 0: Signal /top/s1 is 'W'
# Time [0,20] delta 0: Signal /top/s1 is 'L'
# Time [0,25] delta 0: Signal /top/s1 is 'H'
# Time [0,30] delta 0: Signal /top/s1 is '-'
# Time [0,35] delta 0: Signal /top/s1 is 'U'
# Time [0,40] delta 0: Signal /top/s1 is 'X'
# Time [0,45] delta 0: Signal /top/s1 is '0'
# Time [0,50] delta 0: Signal /top/s1 is '1'
VSIM 2> quit
# Cleaning up...
```

Related Topics

mti_ScheduleWakeup()

mti_Sensitize()

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateProcessWithPriority()

Creates a new VHDL process with a specific priority.

Syntax

process_id = mti_CreateProcessWithPriority(name, func, param, priority)

Arguments

Name	Type	Description
name	char *	The name of the new VHDL process; OPTIONAL - can be NULL
func	mtiVoidFuncPtrT	A pointer to the function that will be executed as the body of the new process
param	void *	A parameter to be passed to the function; OPTIONAL - can be NULL
priority	mtiProcessPriorityT	The priority of the new process: immediate, normal, synch, NBA, or postponed

Return Values

Name	Type	Description
process_id	mtiProcessIdT	A handle to the new VHDL process or NULL if there is an error

Description

mti_CreateProcessWithPriority() creates a new VHDL process with the specified name and priority. If the name is non-NULL, then it appears in the Process window of the simulator; otherwise, it does not. The simulator calls the specified function along with its parameter whenever the process executes. The process executes either at the time specified in a call to mti_ScheduleWakeup() or whenever one of the signals to which it is sensitive changes (mti_Sensitize()).

The priority of the process can be one of the following:

MTI_PROC_IMMEDIATE	All immediate processes run immediately after signal activation (if triggered). If any immediate process activates any signals, then the signals are reevaluated and all immediate processes (if triggered) are run again in the same delta. This cycle continues until no more signals are activated.
MTI_PROC_NORMAL	Normal processes run (when triggered) after all immediate processes have run and settled. They can run once per delta and can schedule events in zero delay.
MTI_PROC_SYNCH	Synchronized processes (when triggered) run after immediate and normal processes, but before NBA processes. They can run once per delta and can schedule events in zero delay.
MTI_PROC_NBA	Non-Blocking Assignment processes (when triggered) run after synchronized processes, but before postponed processes. They can run once per delta and can schedule events in zero delay.
MTI_PROC_POSTPONED	Postponed processes (when triggered) run once at the end of the time step for which they are scheduled after all immediate, normal, synchronized, and NBA processes. They cannot schedule anything in zero delay. (In Verilog, these types of processes are also known as read-only synchronization processes or $monitor() processes.)

If you create the process during elaboration from inside of a foreign architecture instance, then it automatically executes the process once at time zero. If the process is created either after elaboration is complete or from any other context (such as from an initialization function that executes as a result of the loading of a foreign shared library by the -foreign option to vsim), then the simulator does not automatically run the process but must be scheduled or sensitized.

mti_CreateProcessWithPriority() allows you to create a process with an illegal HDL name. This is useful for integrators who provide shared libraries for use by end customers, as this is an easy way to avoid potential name conflicts with HDL processes. We recommend the following naming style:

```
<PREFIX_name>
```

where PREFIX is 3 or 4 characters that denote your software (to avoid name conflicts with other integration software) and *name* is the name of the process. Enclosing the entire name in angle brackets makes it an illegal HDL name. For example, <MTI_foreign_architecture>.

We strongly recommend that you do not use characters in the name that will cause Tcl parsing problems. This includes spaces, the path separator (normally '/' or '.'), square brackets ([]), and dollar signs ($). If you must use these characters, then create an escaped name by putting a backslash (\) at both ends of the name.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>
typedef struct {
  mtiProcessIdT immed_procid[5];
  mtiProcessIdT normal_procid[5];
  mtiProcessIdT synch_procid[5];
  mtiProcessIdT nba_procid[5];
  mtiProcessIdT postponed_procid[5];
  mtiSignalIdT  sig02;
} instanceInfoT;
void scheduleProcesses( instanceInfoT * inst,
int i, mtiDelayT delay )
{
  mti_ScheduleWakeup( inst->immed_procid[i],
    delay );
  mti_ScheduleWakeup( inst->normal_procid[i],
   delay );
  mti_ScheduleWakeup( inst->synch_procid[i],
    delay );
  mti_ScheduleWakeup( inst->nba_procid[i],
      delay );
  mti_ScheduleWakeup( inst->postponed_procid[i],
delay );
}
/* Main test process */
void testProcess( void * param )
{
  instanceInfoT * inst = param;
  mtiInt32T sigval;
  mti_PrintFormatted( "\nTime [%d,%d] delta
%d: testProcess()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
  scheduleProcesses( inst, 0, 0 );
  /* Test immediate activation of immediate
process. */
  sigval = mti_GetSignalValue( inst->sig02
);
  if ( sigval == 0 ) {
    sigval = 1;
  } else {
    sigval = 0;
  }
  mti_SetSignalValue( inst->sig02, (long)sigval
);
}
/* Immediate process sensitive to a signal
in zero-delay */
void sigImmedProc( instanceInfoT * inst )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: sigImmedProc()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
```

```
}
/* Processes scheduled by testProcess() */
void immedProcess1( void * param )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: immedProcess1()\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void normalProcess1( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: normalProcess1():     "
                     "Scheduling processes
ending in 2\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
  scheduleProcesses( inst, 1, 0 );
}
void synchProcess1( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: synchProcess1():      "
                     "Scheduling processes
ending in 3\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
  scheduleProcesses( inst, 2, 0 );
}
void nbaProcess1( instanceInfoT * inst )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: nbaProcess1():        "
                     "Scheduling processes
ending in 4\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
  scheduleProcesses( inst, 3, 0 );
}
void postponedProcess1( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: postponedProcess1(): "
                     "Scheduling processes
ending in 5\n",
                     mti_NowUpper(), mti_Now(),
mti_Delta() );
  scheduleProcesses( inst, 4, 1 );
}
/* Processes scheduled by normalProcess1()
*/
void immedProcess2( void * param )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: immedProcess2()\n",
```

```
                              mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void normalProcess2( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: normalProcess2()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void synchProcess2( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: synchProcess2()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void nbaProcess2( instanceInfoT * inst )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: nbaProcess2()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void postponedProcess2( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: postponedProcess2()\n",
                        mti_NowUpper(), mti_Now(),
mti_Delta() );
}
/* Processes scheduled by synchProcess1()
*/
void immedProcess3( void * param )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: immedProcess3()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void normalProcess3( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: normalProcess3()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void synchProcess3( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: synchProcess3()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
```

```
}
void nbaProcess3( instanceInfoT * inst )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: nbaProcess3()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void postponedProcess3( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: postponedProcess3()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
/* Processes scheduled by nbaProcess1() */
void immedProcess4( void * param )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: immedProcess4()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void normalProcess4( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: normalProcess4()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void synchProcess4( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: synchProcess4()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void nbaProcess4( instanceInfoT * inst )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: nbaProcess4()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void postponedProcess4( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: postponedProcess4()\n",
                      mti_NowUpper(), mti_Now(),
mti_Delta() );
}
/* Processes scheduled by postponedProcess1()
*/
void immedProcess5( void * param )
```

```
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: immedProcess5()\n",
                    mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void normalProcess5( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: normalProcess5()\n",
                    mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void synchProcess5( instanceInfoT * inst
)
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: synchProcess5()\n",
                    mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void nbaProcess5( instanceInfoT * inst )
{
  mti_PrintFormatted( "Time [%d,%d] delta
%d: nbaProcess5()\n",
                    mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void postponedProcess5( instanceInfoT * inst
)
{
mti_PrintFormatted( "Time [%d,%d] delta %d:
postponedProcess5()\n",
                  mti_NowUpper(), mti_Now(),
mti_Delta() );
}
void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}
void initForeign(
  mtiRegionIdT      region,   /* The ID
of the region in which this     */
                                  /* foreign
architecture is instantiated.  */
  char              *param,   /* The last
part of the string in the      */
                                  /* foreign
attribute.                      */
  mtiInterfaceListT *generics, /* A list
of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list
of ports for the foreign model.   */
)
{
  char           * immed_name;
```

```
  char        * normal_name;
  char        * synch_name;
  char        * nba_name;
  char        * postponed_name;
  instanceInfoT * inst;
  int           i;
  mtiProcessIdT  procid;
  mtiSignalIdT   sigid;
  mtiVoidFuncPtrT immed_func;
  mtiVoidFuncPtrT normal_func;
  mtiVoidFuncPtrT synch_func;
  mtiVoidFuncPtrT nba_func;
  mtiVoidFuncPtrT postponed_func;
  inst   = (instanceInfoT *)malloc( sizeof(instanceInfoT)
);
  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcessWithPriority(
"TestProcess", testProcess,
                                           inst,
MTI_PROC_IMMEDIATE );
  mti_Sensitize( procid, sigid, MTI_EVENT
);
  inst->sig02 = mti_FindSignal( "/top/s2"
);
  procid = mti_CreateProcessWithPriority(
"sigImmedProc", sigImmedProc,
                                           inst,
MTI_PROC_IMMEDIATE );
  mti_Sensitize( procid, inst->sig02, MTI_EVENT
);
  for ( i = 0; i < 5; i++ ) {
    switch ( i ) {
      case 0:
        immed_func    = immedProcess1;
        normal_func   = normalProcess1;
        synch_func    = synchProcess1;
        nba_func      = nbaProcess1;
        postponed_func = postponedProcess1;
        immed_name    = "immedProcess1";
        normal_name   = "normalProcess1";
        synch_name    = "synchProcess1";
        nba_name      = "nbaProcess1";
        postponed_name = "postponedProcess1";
        break;
      case 1:
        immed_func    = immedProcess2;
        normal_func   = normalProcess2;
        synch_func    = synchProcess2;
        nba_func      = nbaProcess2;
        postponed_func = postponedProcess2;
        immed_name    = "immedProcess2";
        normal_name   = "normalProcess2";
        synch_name    = "synchProcess2";
        nba_name      = "nbaProcess2";
        postponed_name = "postponedProcess2";
        break;
      case 2:
        immed_func    = immedProcess3;
```

```
           normal_func     = normalProcess3;
           synch_func      = synchProcess3;
           nba_func        = nbaProcess3;
           postponed_func = postponedProcess3;
           immed_name      = "immedProcess3";
           normal_name     = "normalProcess3";
           synch_name      = "synchProcess3";
           nba_name        = "nbaProcess3";
           postponed_name = "postponedProcess3";
           break;
        case 3:
           immed_func      = immedProcess4;
           normal_func     = normalProcess4;
           synch_func      = synchProcess4;
           nba_func        = nbaProcess4;
           postponed_func = postponedProcess4;
           immed_name      = "immedProcess4";
           normal_name     = "normalProcess4";
           synch_name      = "synchProcess4";
           nba_name        = "nbaProcess4";
           postponed_name = "postponedProcess4";
           break;
        case 4:
           immed_func      = immedProcess5;
           normal_func     = normalProcess5;
           synch_func      = synchProcess5;
           nba_func        = nbaProcess5;
           postponed_func = postponedProcess5;
           immed_name      = "immedProcess5";
           normal_name     = "normalProcess5";
           synch_name      = "synchProcess5";
           nba_name        = "nbaProcess5";
           postponed_name = "postponedProcess5";
           break;
        }
        inst->immed_procid[i]  = mti_CreateProcessWithPriority(
immed_name,
                                                     immed_func,
inst,
                                                     MTI_PROC_IMMEDIATE
);
        inst->normal_procid[i] = mti_CreateProcessWithPriority(
normal_name,
                                                     normal_func,
inst,
                                                     MTI_PROC_NORMAL
);
        inst->synch_procid[i]  = mti_CreateProcessWithPriority(
synch_name,
                                                     synch_func,
inst,
                                                     MTI_PROC_SYNCH
);
        inst->nba_procid[i]    = mti_CreateProcessWithPriority(
nba_name,
                                                     nba_func,
inst,
                                                     MTI_PROC_NBA
```

```
);
    inst->postponed_procid[i] = mti_CreateProcessWithPriority
      ( postponed_name, postponed_func, inst,
MTI_PROC_POSTPONED );
    }
  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst
);
  }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  s1 <= not s1 after 5 ns;

  i1 : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl
# 5.4
# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 8
# Time [0,0] delta 0: nbaProcess5()
# Time [0,0] delta 0: synchProcess5()
# Time [0,0] delta 0: normalProcess5()
# Time [0,0] delta 0: immedProcess5()
# Time [0,0] delta 0: nbaProcess4()
# Time [0,0] delta 0: synchProcess4()
# Time [0,0] delta 0: normalProcess4()
# Time [0,0] delta 0: immedProcess4()
# Time [0,0] delta 0: nbaProcess3()
# Time [0,0] delta 0: synchProcess3()
# Time [0,0] delta 0: normalProcess3()
# Time [0,0] delta 0: immedProcess3()
# Time [0,0] delta 0: nbaProcess2()
# Time [0,0] delta 0: synchProcess2()
# Time [0,0] delta 0: normalProcess2()
# Time [0,0] delta 0: immedProcess2()
# Time [0,0] delta 0: nbaProcess1():        Scheduling
processes ending in 4
# Time [0,0] delta 0: synchProcess1():     Scheduling
processes ending in 3
# Time [0,0] delta 0: normalProcess1():    Scheduling
processes ending in 2
# Time [0,0] delta 0: immedProcess1()
# Time [0,0] delta 0: sigImmedProc()
#
# Time [0,0] delta 0: testProcess()
# Time [0,0] delta 0: postponedProcess5()
# Time [0,0] delta 0: postponedProcess4()
# Time [0,0] delta 0: postponedProcess3()
# Time [0,0] delta 0: postponedProcess2()
# Time [0,0] delta 0: postponedProcess1(): Scheduling
processes ending in 5
# Time [0,0] delta 1: sigImmedProc()
# Time [0,0] delta 1: immedProcess4()
# Time [0,0] delta 1: immedProcess3()
# Time [0,0] delta 1: immedProcess2()
# Time [0,0] delta 1: immedProcess1()
# Time [0,0] delta 1: normalProcess4()
# Time [0,0] delta 1: normalProcess3()
# Time [0,0] delta 1: normalProcess2()
# Time [0,0] delta 1: normalProcess1():    Scheduling
processes ending in 2
# Time [0,0] delta 2: immedProcess2()
# Time [0,0] delta 2: normalProcess2()
# Time [0,0] delta 2: synchProcess4()
# Time [0,0] delta 2: synchProcess3()
# Time [0,0] delta 2: synchProcess2()
```

```
# Time [0,0] delta 2: synchProcess1():    Scheduling
processes ending in 3
# Time [0,0] delta 3: immedProcess3()
# Time [0,0] delta 3: normalProcess3()
# Time [0,0] delta 3: synchProcess3()
# Time [0,0] delta 3: nbaProcess4()
# Time [0,0] delta 3: nbaProcess3()
# Time [0,0] delta 3: nbaProcess2()
# Time [0,0] delta 3: nbaProcess1():       Scheduling
processes ending in 4
# Time [0,0] delta 4: immedProcess4()
# Time [0,0] delta 4: normalProcess4()
# Time [0,0] delta 4: synchProcess4()
# Time [0,0] delta 4: nbaProcess4()
# Time [0,0] delta 4: postponedProcess4()
# Time [0,0] delta 4: postponedProcess3()
# Time [0,0] delta 4: postponedProcess2()
# Time [0,0] delta 4: postponedProcess1(): Scheduling
processes ending in 5
# Time [0,1] delta 0: immedProcess5()
# Time [0,1] delta 0: normalProcess5()
# Time [0,1] delta 0: synchProcess5()
# Time [0,1] delta 0: nbaProcess5()
# Time [0,1] delta 0: postponedProcess5()
#
# Time [0,5] delta 0: testProcess()
# Time [0,5] delta 0: sigImmedProc()
# Time [0,5] delta 1: immedProcess1()
# Time [0,5] delta 1: normalProcess1():    Scheduling
processes ending in 2
# Time [0,5] delta 2: immedProcess2()
# Time [0,5] delta 2: normalProcess2()
# Time [0,5] delta 2: synchProcess2()
# Time [0,5] delta 2: synchProcess1():     Scheduling
processes ending in 3
# Time [0,5] delta 3: immedProcess3()
# Time [0,5] delta 3: normalProcess3()
# Time [0,5] delta 3: synchProcess3()
# Time [0,5] delta 3: nbaProcess3()
# Time [0,5] delta 3: nbaProcess2()
# Time [0,5] delta 3: nbaProcess1():       Scheduling
processes ending in 4
# Time [0,5] delta 4: immedProcess4()
# Time [0,5] delta 4: normalProcess4()
# Time [0,5] delta 4: synchProcess4()
# Time [0,5] delta 4: nbaProcess4()
# Time [0,5] delta 4: postponedProcess4()
# Time [0,5] delta 4: postponedProcess3()
# Time [0,5] delta 4: postponedProcess2()
# Time [0,5] delta 4: postponedProcess1(): Scheduling
processes ending in 5
# Time [0,6] delta 0: immedProcess5()
# Time [0,6] delta 0: normalProcess5()
# Time [0,6] delta 0: synchProcess5()
# Time [0,6] delta 0: nbaProcess5()
# Time [0,6] delta 0: postponedProcess5()
VSIM 2> quit
# Cleaning up...
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateRealType()

Creates a real type.

Syntax

type_id = mti_CreateRealType()

Arguments

None

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the new real type

Description

mti_CreateRealType() creates a new type ID that describes a VHDL real type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef struct {
  mtiSignalIdT sigid;
  mtiDriverIdT drvid;
} instanceInfoT;

void driveSignal( void * param )
{
  char         * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  double         sigval;

  (void)mti_GetSignalValueIndirect( inst->sigid, &sigval );
  region_name = mti_GetRegionFullName( mti_GetSignalRegion( inst->sigid ) );

  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %g\n",
                     mti_NowUpper(), mti_Now(), mti_Delta(),
                     region_name, mti_GetSignalName( inst->sigid), sigval );

  sigval = sigval + 1.5;
  mti_ScheduleDriver( inst->drvid, (long)&sigval, 5, MTI_INERTIAL );

  mti_VsimFree( region_name );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instanceInfoT * inst;
  mtiProcessIdT   procid;
  mtiTypeIdT      real_type;

  inst        = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
  real_type   = mti_CreateRealType();
  inst->sigid = mti_CreateSignal( "mySig", region, real_type );
  inst->drvid = mti_CreateDriver( inst->sigid );
  procid      = mti_CreateProcess( "mySigDriver", driveSignal, inst );
  mti_Sensitize( procid, inst->sigid, MTI_EVENT );
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid, procid );
```

```
    mti_AddQuitCB( cleanupCallback, inst );
    mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
entity top is
end top;

architecture a of top is
   signal s1 : bit := '0';
begin
   s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,0] delta 1: Signal /top/mysig is 0
# Time [0,5] delta 0: Signal /top/mysig is 1.5
# Time [0,10] delta 0: Signal /top/mysig is 3
# Time [0,15] delta 0: Signal /top/mysig is 4.5
# Time [0,20] delta 0: Signal /top/mysig is 6
# Time [0,25] delta 0: Signal /top/mysig is 7.5
# Time [0,30] delta 0: Signal /top/mysig is 9
# Time [0,35] delta 0: Signal /top/mysig is 10.5
# Time [0,40] delta 0: Signal /top/mysig is 12
# Time [0,45] delta 0: Signal /top/mysig is 13.5
# Time [0,50] delta 0: Signal /top/mysig is 15
VSIM 2> quit
# Cleaning up...
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateRegion()

Creates a new VHDL region.

Syntax

region_id = mti_CreateRegion(parent, name)

Arguments

Name	Type	Description
parent	mtiRegionIdT	A handle to the parent region under which the new region is to be placed; OPTIONAL - can be NULL
name	char *	The name of the new region; OPTIONAL - can be NULL

Return Values

Name	Type	Description
region_id	mtiRegionIdT	A handle to the new region or NULL if there is an error

Description

mti_CreateRegion() creates a new region with the specified name under the specified parent region. The simulator converts the name to lower case unless the parent is VHDL and retains the case if the parent is Verilog or SystemC. If the name is NULL, then the region is hidden. The simulator does not connect the new region to the design hierarchy, if the parent region is NULL.

You can create the new region below a VHDL, Verilog, or SystemC region, where the new region is of type accForeign and of fulltype accShadow (refer to acc_vhdl.h).

If you create a region with no name or with no parent, you must save the returned handle to the region as there is no way to find the region by name or by traversing the design with the region traversal functions.

mti_CreateRegion() allows you to create a region with an illegal HDL name. This is useful for integrators who provide shared libraries for use by end customers, as this is an easy way to avoid potential name conflicts with HDL regions. We recommend the following naming style:

```
<PREFIX_name>
```

where PREFIX is 3 or 4 characters that denote your software (to avoid name conflicts with other integration software) and *name* is the name of the region. Enclosing the entire name in angle brackets makes it an illegal HDL name. For example, <MTI_region>.

We strongly recommend that you do not use characters in the name that will cause Tcl parsing problems. This includes spaces, the path separator (normally '/' or '.'), square brackets ([]), angle brackets (< >), and dollar signs ($). If you must use these characters, then create an escaped name by putting a backslash (\) at both ends of the name.

Examples

FLI code

```c
#include <acc_user.h>
#include <acc_vhdl.h>
#include <mti.h>

void printRegionInfo( mtiRegionIdT regid, int indent )
{
  char       * regkind;
  mtiRegionIdT subreg;

  switch ( mti_GetRegionKind( regid ) ) {
    case accArchitecture:
      regkind = "Architecture";
      break;
    case accForeign:
      regkind = "Foreign";
      break;
    case accModule:
      regkind = "Module";
      break;
    case accPackage:
       regkind = "Package";
       break;
    default:
      regkind = "Unknown";
      break;
  }
  mti_PrintFormatted( "%*cRegion %s : %s\n", indent, ' ',
                      mti_GetRegionName( regid ), regkind );
  indent += 2;
  for ( subreg = mti_FirstLowerRegion( regid ); subreg;
        subreg = mti_NextRegion( subreg ) ) {
    printRegionInfo( subreg, indent );
  }
}

void loadDone( void * param )
{
  mtiRegionIdT foreign_region = (mtiRegionIdT)param;
  mtiRegionIdT parent;
  mtiRegionIdT regid;

  (void) mti_CreateRegion( foreign_region, "reg_under_for_arch_post_elab" );

  parent = mti_HigherRegion( foreign_region );
  (void) mti_CreateRegion( parent, "region_under_parent_post_elab" );

  for (regid = mti_GetTopRegion(); regid; regid = mti_NextRegion( regid )) {
    printRegionInfo( regid, 1 );
  }
}

void initForeign(
  mtiRegionIdT       region,    /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
```

```
        char              *param,    /* The last part of the string in the    */
                                     /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
   )
   {
     mtiRegionIdT parent;
     (void) mti_CreateRegion( region, "region_under_foreign_arch" );

     parent = mti_HigherRegion( region );
     (void) mti_CreateRegion( parent, "region_under_parent" );

     (void) mti_CreateRegion( region, 0 );  /* Region with no name */

     mti_AddLoadDoneCB( loadDone, region );
   }
```

HDL code

```
   entity for_model is
   end for_model;

   architecture a of for_model is
     attribute foreign of a : architecture is "initForeign for_model.sl";
   begin
   end a;

   library ieee;
   use ieee.std_logic_1164.all;

   entity top is
   end top;

   architecture a of top is

     signal s1 : std_logic := '0';

     component for_model is
     end component;

     for all : for_model use entity work.for_model(a);

   begin
     i1 : for_model;
   end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#  Region top : Architecture
#    Region region_under_parent_post_elab : Foreign
#    Region i1 : Architecture
#      Region reg_under_for_arch_post_elab : Foreign
#      Region region_under_foreign_arch : Foreign
#    Region region_under_parent : Foreign
#  Region standard : Package
#  Region std_logic_1164 : Package
VSIM 1> run 5
VSIM 2> quit
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateScalarType()

Creates a scalar type.

Syntax

type_id = mti_CreateScalarType(left, right)

Arguments

Name	Type	Description
left	mtiInt32T	The left-most value of the new scalar type
right	mtiInt32T	The right-most value of the new scalar type

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the new scalar type

Description

mti_CreateScalarType() creates a new type ID that describes a VHDL scalar (integer) type whose value range is determined by the specified left and right values.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef struct {
  mtiSignalIdT sigid;
  mtiDriverIdT drvid;
} instanceInfoT;

void driveSignal( void * param )
{
  char        * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;

  sigval = mti_GetSignalValue( inst->sigid );
  region_name = mti_GetRegionFullName( mti_GetSignalRegion( inst->sigid ) );

  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %d\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      region_name, mti_GetSignalName( inst->sigid), sigval );

  sigval = sigval + 2;
  mti_ScheduleDriver( inst->drvid, sigval, 5, MTI_INERTIAL );

  mti_VsimFree( region_name );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instanceInfoT * inst;
  mtiProcessIdT   procid;
  mtiTypeIdT      scalar_type;

  inst        = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
  scalar_type = mti_CreateScalarType( 0, 100 );
  inst->sigid = mti_CreateSignal( "mySig", region, scalar_type );
  inst->drvid = mti_CreateDriver( inst->sigid );
  procid      = mti_CreateProcess( "mySigDriver", driveSignal, inst );
  mti_Sensitize( procid, inst->sigid, MTI_EVENT );
  mti_ScheduleWakeup( procid, 0 );
  mti_SetDriverOwner( inst->drvid, procid );
```

```
    mti_AddQuitCB( cleanupCallback, inst );
    mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
entity top is
end top;

architecture a of top is
   signal s1 : bit := '0';
begin
   s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,0] delta 1: Signal /top/mysig is 0
# Time [0,5] delta 0: Signal /top/mysig is 2
# Time [0,10] delta 0: Signal /top/mysig is 4
# Time [0,15] delta 0: Signal /top/mysig is 6
# Time [0,20] delta 0: Signal /top/mysig is 8
# Time [0,25] delta 0: Signal /top/mysig is 10
# Time [0,30] delta 0: Signal /top/mysig is 12
# Time [0,35] delta 0: Signal /top/mysig is 14
# Time [0,40] delta 0: Signal /top/mysig is 16
# Time [0,45] delta 0: Signal /top/mysig is 18
# Time [0,50] delta 0: Signal /top/mysig is 20
VSIM 2> quit
# Cleaning up...
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateSignal()

Creates a new VHDL signal.

Syntax

signal_id = mti_CreateSignal(name, region, type)

Arguments

Name	Type	Description
name	char *	The name of the new VHDL signal; OPTIONAL - can be NULL
region	mtiRegionIdT	The design region into which the new signal is to be placed
type	mtiTypeIdT	The type of the new signal

Return Values

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the new VHDL signal or NULL if there is an error

Description

mti_CreateSignal() creates a new VHDL signal of the specified type in the specified region. If the name is not NULL, then the signal will appear in the Signals window of the simulator.

The simulator converts all signal names that do not start and end with a '\' to lower case. It also treats signal names starting and ending with '\' as VHDL extended identifiers and uses them, unchanged.

You can create the new signal within a SystemC region.

mti_CreateSignal() allows you to create a signal with an illegal HDL name. This is useful for integrators who provide shared libraries for use by end customers, as this is an easy way to avoid potential name conflicts with HDL signals. We recommend the following naming style:

```
<PREFIX_name>
```

where PREFIX is 3 or 4 characters that denote your software (to avoid name conflicts with other integration software) and *name* is the name of the signal. Enclosing the entire name in angle brackets makes it an illegal HDL name. For example, <MTI_siga>.

We strongly recommend that you do not use characters in the name that will cause Tcl parsing problems. This includes spaces, the path separator (normally '/' or '.'), square brackets ([]), and dollar signs ($). If you must use these characters, then create an escaped name by putting a backslash (\) at both ends of the name.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} mySigType;

char *std_logic_lits[9] =
{ "'U'", "'X'", "'0'", "'1'", "'Z'", "'W'", "'L'", "'H'", "'-'" };

typedef struct {
  mtiSignalIdT sigid1;
  mtiSignalIdT sigid2;
  mtiSignalIdT sigid3;
  mtiDriverIdT drvid1;
  mtiDriverIdT drvid2;
  mtiDriverIdT drvid3;
} instanceInfoT;

/* This function inverts mySig1 every 5 ns. */
void driveSignal1( void * param )
{
  char         * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T      sigval;

  sigval = mti_GetSignalValue( inst->sigid1 );

  switch ( sigval ) {
   case STD_LOGIC_U:
    mti_ScheduleDriver( inst->drvid1, STD_LOGIC_0, 0, MTI_INERTIAL );
    break;
   case STD_LOGIC_0:
    mti_ScheduleDriver( inst->drvid1, STD_LOGIC_1, 5, MTI_INERTIAL );
    break;
   case STD_LOGIC_1:
    mti_ScheduleDriver( inst->drvid1, STD_LOGIC_0, 5, MTI_INERTIAL );
    break;
   case STD_LOGIC_X:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
    mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is UNKNOWN\n",
                    mti_NowUpper(), mti_Now(), mti_Delta(),
                    region_name, mti_GetSignalName( inst->sigid1 ) );
    mti_VsimFree( region_name );
    break;
   default:
```

```
        region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
        mti_PrintFormatted( "Time [%d,%d] delta %d: "
                            "Unexpected value %d on signal %s/%s\n",
                            mti_NowUpper(), mti_Now(), mti_Delta(),
                            sigval, region_name,
                            mti_GetSignalName( inst->sigid1 ) );
        mti_VsimFree( region_name );
        break;
    }
}

/* This function inverts mySig2 every 10 ns. */
void driveSignal2( void * param )
{
  char         * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;

  sigval = mti_GetSignalValue( inst->sigid2 );

  switch ( sigval ) {
   case STD_LOGIC_U:
    mti_ScheduleDriver( inst->drvid2, STD_LOGIC_0, 0, MTI_INERTIAL );
    break;
   case STD_LOGIC_0:
    mti_ScheduleDriver( inst->drvid2, STD_LOGIC_1, 10, MTI_INERTIAL );
    break;
   case STD_LOGIC_1:
    mti_ScheduleDriver( inst->drvid2, STD_LOGIC_0, 10, MTI_INERTIAL );
    break;
   case STD_LOGIC_X:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
    mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is UNKNOWN\n",
                        mti_NowUpper(), mti_Now(), mti_Delta(),
                        region_name, mti_GetSignalName( inst->sigid2 ) );
    mti_VsimFree( region_name );
    break;
   default:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
    mti_PrintFormatted( "Time [%d,%d] delta %d: "
                        "Unexpected value %d on signal %s/%s\n",
                        mti_NowUpper(), mti_Now(), mti_Delta(),
                        sigval, region_name,
                        mti_GetSignalName( inst->sigid2 ) );
    mti_VsimFree( region_name );
    break;
  }
}

/* This function drives mySig3 with the values of mySig1 and mySig2. */
void driveSignal3( void * param )
{
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval1;
  mtiInt32T       sigval2;

  sigval1 = mti_GetSignalValue( inst->sigid1 );
  sigval2 = mti_GetSignalValue( inst->sigid2 );
```

```
        if ( sigval1 == sigval2 ) {
            mti_ScheduleDriver( inst->drvid3, sigval1, 0, MTI_INERTIAL );
        } else {
            mti_ScheduleDriver( inst->drvid3, STD_LOGIC_X, 0, MTI_INERTIAL );
        }
    }

    void cleanupCallback( void * param )
    {
      mti_PrintMessage( "Cleaning up...\n" );
      free( param );
    }

    void initForeign(
      mtiRegionIdT         region,    /* The ID of the region in which this     */
                                      /* foreign architecture is instantiated.  */
      char                *param,     /* The last part of the string in the     */
                                      /* foreign attribute.                     */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
    )
    {
      instanceInfoT * inst;
      mtiProcessIdT   procid;
      mtiTypeIdT      enum_type;

      inst       = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
      enum_type  = mti_CreateEnumType( 1, 9, std_logic_lits );

      inst->sigid1 = mti_CreateSignal( "mySig1", region, enum_type );
      inst->drvid1 = mti_CreateDriver( inst->sigid1 );
      procid       = mti_CreateProcess( "mySig1Driver", driveSignal1, inst );
      mti_Sensitize( procid, inst->sigid1, MTI_EVENT );
      mti_ScheduleWakeup( procid, 0 );
      mti_SetDriverOwner( inst->drvid1, procid );

      inst->sigid2 = mti_CreateSignal( "mySig2", region, enum_type );
      inst->drvid2 = mti_CreateDriver( inst->sigid2 );
      procid       = mti_CreateProcess( "mySig2Driver", driveSignal2, inst );
      mti_Sensitize( procid, inst->sigid2, MTI_EVENT );
      mti_ScheduleWakeup( procid, 0 );
      mti_SetDriverOwner( inst->drvid2, procid );

      inst->sigid3 = mti_CreateSignal( "mySig3", region, enum_type );
      inst->drvid3 = mti_CreateDriver( inst->sigid3 );
      procid       = mti_CreateProcess( "mySig3Driver", driveSignal3, inst );
      mti_Sensitize( procid, inst->sigid1, MTI_EVENT );
      mti_Sensitize( procid, inst->sigid2, MTI_EVENT );
      mti_ScheduleWakeup( procid, 0 );
      mti_SetDriverOwner( inst->drvid3, procid );

      mti_AddQuitCB( cleanupCallback, inst );
      mti_AddRestartCB( cleanupCallback, inst );
    }
```

HDL code

```
entity top is
end top;

architecture a of top is
  signal s1 : bit := '0';
begin
  s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 0
VSIM 2> examine mySig1 mySig2 mySig3
# 0 0 0
VSIM 3> run 5
VSIM 4> examine mySig1 mySig2 mySig3
# 1 0 X
VSIM 5> run 5
VSIM 6> examine mySig1 mySig2 mySig3
# 0 1 X
VSIM 7> run 5
VSIM 8> examine mySig1 mySig2 mySig3
# 1 1 1
VSIM 9> quit
# Cleaning up...
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_CreateTimeType()

Creates a time type.

Syntax

type_id = mti_CreateTimeType()

Arguments

None

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the new time type

Description

mti_CreateTimeType() creates a new type ID that describes a VHDL time type.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef struct {
    mtiSignalIdT sigid;
    mtiDriverIdT drvid;
    mtiTime64T   sigval;
    mtiTypeIdT   time_type;
} instanceInfoT;

void driveSignal( void * param )
{
    char          * region_name;
    char          * curr_time_str;
    instanceInfoT * inst = (instanceInfoT*)param;
    mtiTime64T      curr_time;

    region_name = mti_GetRegionFullName( mti_GetSignalRegion( inst->sigid ) );

    curr_time_str = mti_Image( mti_NowIndirect(&curr_time), inst->time_type );
    mti_PrintFormatted( "Time %s delta %d: Signal %s/%s is %s\n",
                        curr_time_str, mti_Delta(),
                        region_name, mti_GetSignalName( inst->sigid),
                        mti_SignalImage(inst->sigid) );

    MTI_TIME64_ASGN( inst->sigval, MTI_TIME64_HI32(inst->sigval),
                     MTI_TIME64_LO32(inst->sigval) + 1 );
    mti_ScheduleDriver( inst->drvid, (long)&(inst->sigval), 5, MTI_INERTIAL );

    mti_VsimFree( region_name );
}

void initForeign(
    mtiRegionIdT       region,   /* The ID of the region in which this     */
                                 /* foreign architecture is instantiated.  */
    char              *param,    /* The last part of the string in the     */
                                 /* foreign attribute.                     */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model. */
)
{
    instanceInfoT * inst;
    mtiProcessIdT   procid;

    inst            = (instanceInfoT *)mti_Malloc( sizeof(instanceInfoT) );
    inst->time_type = mti_CreateTimeType();
    inst->sigid     = mti_CreateSignal( "mySig", region, inst->time_type );
    inst->drvid     = mti_CreateDriver( inst->sigid );
    procid          = mti_CreateProcess( "mySigDriver", driveSignal, inst );
    mti_SetDriverOwner( inst->drvid, procid );
    mti_Sensitize( procid, inst->sigid, MTI_EVENT );
    mti_ScheduleWakeup( procid, 0 );
}
```

HDL code

```
entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

begin

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time {0 ns} delta 1: Signal /top/mysig is {0 ns}
# Time {5 ns} delta 0: Signal /top/mysig is {1 ns}
# Time {10 ns} delta 0: Signal /top/mysig is {2 ns}
# Time {15 ns} delta 0: Signal /top/mysig is {3 ns}
# Time {20 ns} delta 0: Signal /top/mysig is {4 ns}
# Time {25 ns} delta 0: Signal /top/mysig is {5 ns}
# Time {30 ns} delta 0: Signal /top/mysig is {6 ns}
# Time {35 ns} delta 0: Signal /top/mysig is {7 ns}
# Time {40 ns} delta 0: Signal /top/mysig is {8 ns}
# Time {45 ns} delta 0: Signal /top/mysig is {9 ns}
# Time {50 ns} delta 0: Signal /top/mysig is {10 ns}
VSIM 2> quit
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_Delta()

Gets the simulator iteration count for the current time step.

Syntax

delta = mti_Delta()

Arguments

None

Return Values

Name	Type	Description
delta	mtiUInt32T	The simulator iteration count for the current time step

Description

mti_Delta() returns the simulator iteration count for the current time step.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}

void monitorSignal( void * param )
{
  char          * region_name;
  mtiSignalIdT    sigid = (mtiSignalIdT)param;
  mtiInt32T       sigval;

  region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
  sigval = mti_GetSignalValue( sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      region_name, mti_GetSignalName( sigid ),
                      convertStdLogicValue( sigval ) );
  mti_VsimFree( region_name );
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
  char               *param,    /* The last part of the string in the    */
                                /* foreign attribute.                    */
```

```
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
)
{
  mtiProcessIdT   procid;
  mtiSignalIdT    sigid;

  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "sigMonitor", monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is
  signal s1 : std_logic := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  p1 : process
  begin
    s1 <= '1';
    wait for 5 ns;
    s1 <= '0';
    wait for 0 ns;
    s1 <= '1';
    wait for 0 ns;
    s1 <= '0';
    wait for 5 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
# Time [0,0] delta 0: Signal /top/s1 is '0'
# Time [0,0] delta 1: Signal /top/s1 is '1'
# Time [0,5] delta 1: Signal /top/s1 is '0'
# Time [0,5] delta 2: Signal /top/s1 is '1'
# Time [0,5] delta 3: Signal /top/s1 is '0'
# Time [0,10] delta 1: Signal /top/s1 is '1'
# Time [0,15] delta 1: Signal /top/s1 is '0'
# Time [0,15] delta 2: Signal /top/s1 is '1'
# Time [0,15] delta 3: Signal /top/s1 is '0'
# Time [0,20] delta 1: Signal /top/s1 is '1'
VSIM 2> quit
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_Desensitize()

Desensitizes a VHDL process to the VHDL or SystemC signals to which it is sensitive.

Syntax

mti_Desensitize(proc)

Arguments

Name	Type	Description
proc	mtiProcessIdT	A handle to a VHDL process

Return Values

Nothing

Description

mti_Desensitize() disconnects a process from the signals to which it is sensitive. You can then re-sensitize (mti_Sensitize()) or schedule (mti_ScheduleWakeup()) the process.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  mtiSignalIdT  sigid;
  mtiProcessIdT procid;
} instanceInfoT;

char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}

void monitorSignal( void * param )
{
  char          * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;

  region_name = mti_GetRegionFullName( mti_GetSignalRegion( inst->sigid ) );
  sigval = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %s\n",
                    mti_NowUpper(), mti_Now(), mti_Delta(),
                    region_name, mti_GetSignalName( inst->sigid ),
                    convertStdLogicValue( sigval ) );
  if ( mti_Now() >= 20 ) {
      mti_PrintFormatted( "  Desensitizing process %s\n",
                      mti_GetProcessName( inst->procid ) );
```

```
        mti_Desensitize( inst->procid );
    }
    mti_VsimFree( region_name );
}

void cleanupCallback( void * param )
{
    mti_PrintMessage( "Cleaning up...\n" );
    free( param );
}

void initForeign(
    mtiRegionIdT      region,   /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
    char             *param,    /* The last part of the string in the    */
                                /* foreign attribute.                    */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
    instanceInfoT * inst;

    inst        = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
    inst->sigid  = mti_FindSignal( "/top/s1" );
    inst->procid = mti_CreateProcess( "sigMonitor", monitorSignal,inst );
    mti_Sensitize( inst->procid, inst->sigid, MTI_EVENT );
    mti_AddQuitCB( cleanupCallback, inst );
    mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  p1 : process
  begin
    s1 <= '1';
    wait for 5 ns;
    s1 <= '0';
    wait for 0 ns;
    s1 <= '1';
    wait for 0 ns;
    s1 <= '0';
    wait for 5 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 30
# Time [0,0] delta 0: Signal /top/s1 is '0'
# Time [0,0] delta 1: Signal /top/s1 is '1'
# Time [0,5] delta 1: Signal /top/s1 is '0'
# Time [0,5] delta 2: Signal /top/s1 is '1'
# Time [0,5] delta 3: Signal /top/s1 is '0'
# Time [0,10] delta 1: Signal /top/s1 is '1'
# Time [0,15] delta 1: Signal /top/s1 is '0'
# Time [0,15] delta 2: Signal /top/s1 is '1'
# Time [0,15] delta 3: Signal /top/s1 is '0'
# Time [0,20] delta 1: Signal /top/s1 is '1'
#   Desensitizing process sigMonitor
VSIM 2> run 10
VSIM 3> quit
# Cleaning up...
```

Related Topics

mti_Free()

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_Exit()

Terminates the simulation immediately, returning the specified exit_status.

Syntax

mti_Exit(exit_status)

Arguments

Name	Type	Description
exit_status	int	Return status of the program

Return Values

Exit status information.

Description

Use this call to define an exit code different from the standard simulator exit codes. this will ensure that simulation fully exits and correctly flushes to disk any simulation output, such as logs, waveform and coverage databases, and so on.

This is opposed to a bare *exit()* call, which, when used in place of mti_Exit(), could prevent the simulator from generating all of the desired information and writing it to these files.

mti_FatalError()

Requests the simulator to halt with a fatal error.

Syntax

mti_FatalError()

Arguments

None

Return Values

Nothing

Description

mti_FatalError() causes the simulator to immediately halt the simulation and issue an assertion message with the text "** Fatal: Foreign module requested halt". A call to mti_FatalError() does not return control to the caller. You cannot continue the simulation after being halted with mti_FatalError().

Examples

FLI code

```c
#include <mti.h>

typedef enum {
  STD_LOGIC_U,       /* 'U' */
  STD_LOGIC_X,       /* 'X' */
  STD_LOGIC_0,       /* '0' */
  STD_LOGIC_1,       /* '1' */
  STD_LOGIC_Z,       /* 'Z' */
  STD_LOGIC_W,       /* 'W' */
  STD_LOGIC_L,       /* 'L' */
  STD_LOGIC_H,       /* 'H' */
  STD_LOGIC_D        /* '-' */
} StdLogicT;

void monitorSignal( void * param )
{
  mtiSignalIdT sigid = (mtiSignalIdT)param;

  switch ( mti_GetSignalValue( sigid ) ) {
    case STD_LOGIC_X:
    case STD_LOGIC_W:
        mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s is UNKNOWN\n",
                            mti_NowUpper(), mti_Now(), mti_Delta(),
                            mti_GetSignalName( sigid ) );
        mti_FatalError();
        break;
    default:
        break;
  }
}
void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;

  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "SignalMonitor", monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

begin

  p1 : process
  begin
    c1 : case s1 is
      when 'U' => s1 <= 'X' after 5 ns;
      when 'X' => s1 <= '0' after 5 ns;
      when '0' => s1 <= '1' after 5 ns;
      when '1' => s1 <= 'Z' after 5 ns;
      when 'Z' => s1 <= 'W' after 5 ns;
      when 'W' => s1 <= 'L' after 5 ns;
      when 'L' => s1 <= 'H' after 5 ns;
      when 'H' => s1 <= '-' after 5 ns;
      when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    wait for 5 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 20
# Time [0,15] delta 0: Signal s1 is UNKNOWN
# ** Fatal: Foreign module requested halt.
#    Time: 15 ns  Iteration: 0  Foreign Process: /top/SignalMonitor File:
Foreign
# Fatal error at  line 0
#
VSIM 2> cont
# Cannot continue because of fatal error.
VSIM 3> quit
```

Related Topics

mti_Free()

---

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_FindDriver()

Determines if a VHDL signal has any drivers on it.

Syntax

driver_id = mti_FindDriver(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the VHDL signal

Return Values

Name	Type	Description
driver_id	mtiDriverIdT	A handle to a driver of the specified signal or NULL if there is an error, if no drivers are found for a scalar signal, or if any element of an array signal does not have a driver

Description

mti_FindDriver() returns a handle to a driver of the specified signal. If the simulator finds no drivers a scalar signal or if any element of an array signal does not have a driver, then it returns NULL. You can free the returned handle with mti_Free(). The driver remains in effect even when you free the handle.

mti_FindDriver() essentially returns the first driver in the signal's driver list. You cannot tell which driver it is, so we do not recommend that you use this driver to drive values from an FLI application. Use mti_FindDriver() simply to determine whether a signal has any drivers.

Examples

FLI code

```
#include <mti.h>

void loadDoneCB( void * param )
{
  mtiDriverIdT  drvid;
  mtiSignalIdT  sigid;

  sigid = mti_FindSignal( "/top/s1" );
  drvid = mti_FindDriver( sigid );
  mti_PrintFormatted( "Driver %sfound for /top/s1\n", drvid ? "" : "not " );

  sigid = mti_FindSignal( "/top/s2" );
  drvid = mti_FindDriver( sigid );
  mti_PrintFormatted( "Driver %sfound for /top/s2\n", drvid ? "" : "not " );

  sigid = mti_FindSignal( "/top/s3" );
  drvid = mti_FindDriver( sigid );
  mti_PrintFormatted( "Driver %sfound for /top/s3\n", drvid ? "" : "not " );

  sigid = mti_FindSignal( "/top/s4" );
  drvid = mti_FindDriver( sigid );
  mti_PrintFormatted( "Driver %sfound for /top/s4\n", drvid ? "" : "not " );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';
  signal s2 : std_logic := '0';
  signal s3 : std_logic_vector( 3 downto 0 ) := "0110";
  signal s4 : std_logic_vector( 3 downto 0 ) := "1010";

begin

  s4 <= not s4 after 5 ns;

  p1 : process
  begin
    c1 : case s1 is
      when 'U' => s1 <= 'X' after 5 ns;
      when 'X' => s1 <= '0' after 5 ns;
      when '0' => s1 <= '1' after 5 ns;
      when '1' => s1 <= 'Z' after 5 ns;
      when 'Z' => s1 <= 'W' after 5 ns;
      when 'W' => s1 <= 'L' after 5 ns;
      when 'L' => s1 <= 'H' after 5 ns;
      when 'H' => s1 <= '-' after 5 ns;
      when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    s3(3) <= not s3(3) after 5 ns;
    wait for 5 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
# Driver found for /top/s1
# Driver not found for /top/s2
# Driver not found for /top/s3
# Driver found for /top/s4
VSIM 1> run 10
VSIM 2> quit
```

Related Topics

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_FindPort()

Finds a VHDL or SystemC port signal in a port interface list.

Syntax

signal_id = mti_FindPort(list, name)

Arguments

Name	Type	Description
list	mtiInterfaceListT *	A pointer to a list of interface objects
name	char *	The name of the signal to be found in the list

Return Values

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC port signal or NULL if the signal is not found

Description

mti_FindPort() searches linearly through the specified interface list and returns a handle to the VHDL or SystemC port signal whose name matches the specified name. The search is not case-sensitive.

Examples

FLI code

```c
#include <mti.h>

typedef struct {
  mtiProcessIdT procid;
  mtiSignalIdT  bitsig;
  mtiSignalIdT  intsig;
  mtiSignalIdT  realsig;
} instanceInfoT;

void checkValues( void * param )
{
  double        real_val;
  instanceInfoT * inst = (instanceInfoT*)param;

  mti_PrintFormatted( "Time [%d,%d] delta %d:\n",
                      mti_NowUpper(), mti_Now(), mti_Delta() );
  mti_PrintFormatted( "  %s = %d\n",
                      mti_GetSignalName( inst->bitsig ),
                      mti_GetSignalValue( inst->bitsig ) );
  mti_PrintFormatted( "  %s = %d\n",
                      mti_GetSignalName( inst->intsig ),
                      mti_GetSignalValue( inst->intsig ) );
  (void) mti_GetSignalValueIndirect( inst->realsig, &real_val );
  mti_PrintFormatted( "  %s = %g\n",
                      mti_GetSignalName( inst->realsig ), real_val );
  mti_ScheduleWakeup( inst->procid, 5 );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instanceInfoT * inst;

  inst         = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
  inst->intsig  = mti_FindPort( ports, "PORT2" );
  inst->bitsig  = mti_FindPort( ports, "p1" );
  inst->realsig = mti_FindPort( ports, "rPort" );

  inst->procid = mti_CreateProcess( "ValueChecker", checkValues, inst );

  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst );
```

```
      }
```

HDL code

```
entity for_model is
  port ( p1    : bit;
         port2 : integer;
         rport : real
       );
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : integer := 42;
  signal s3 : real := 1.57;

  component for_model is
    port ( p1    : bit;
           port2 : integer;
           rport : real
         );
  end component;

  for all : for_model use entity work.for_model(a);

begin
  i1 : for_model
       port map ( s1, s2, s3 );

  s1 <= not s1 after 5 ns;
  s2 <= s2 + 1 after 5 ns;
  s3 <= s3 + 1.5 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,0] delta 0:
#   s1 = 0
#   s2 = 42
#   s3 = 1.57
# Time [0,5] delta 0:
#   s1 = 1
#   s2 = 43
#   s3 = 3.07
# Time [0,10] delta 0:
#   s1 = 0
#   s2 = 44
#   s3 = 4.57
# Time [0,15] delta 0:
#   s1 = 1
#   s2 = 45
#   s3 = 6.07
VSIM 2> quit
# Cleaning up...
```

mti_GetSignalName() returns the name of the top-level signal connected to each port because of standard simulator optimization that collapses hierarchical port connections wherever possible.

Related Topics

mti_VsimFree()

mti_NextRegion()

mti_GetSignalSubelements()

mti_TickLength()

mti_FindProjectEntry()

Gets the value of an entry in the project (.ini) file.

Syntax

value = mti_FindProjectEntry(section, name, expand)

Arguments

Name	Type	Description
section	char *	The name of the section in the project file in which the entry resides
name	char *	The name of the entry
int	expand	If this parameter is non-zero, then environment variables in the entry are expanded; otherwise they are not

Return Values

Name	Type	Description
value	char *	The value of the specified entry or NULL if the entry is not found

Description

mti_FindProjectEntry() returns the value of the specified entry from the specified section of the project file (*modelsim.ini*). Expansion of environment variables in the entry's value is controlled by the expand parameter. The comparison against the section and name strings is not case-sensitive.

The caller is responsible for freeing the returned pointer with mti_VsimFree().

Examples

FLI code

```
#include <mti.h>

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this      */
                              /* foreign architecture is instantiated.   */
  char             *param,    /* The last part of the string in the      */
                              /* foreign attribute.                      */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  char * entry;

  entry = mti_FindProjectEntry( "myconfig", "myentry", 0 );
  mti_PrintFormatted( "[myconfig] myentry = %s\n", entry );
  mti_VsimFree( entry );

  entry = mti_FindProjectEntry( "myconfig", "myentry", 1 );
  mti_PrintFormatted( "[myconfig] myentry = %s\n", entry );
  mti_VsimFree( entry );

  entry = mti_FindProjectEntry( "library", "std", 0 );
  mti_PrintFormatted( "[Library] std = %s\n", entry );
  mti_VsimFree( entry );

  entry = mti_FindProjectEntry( "VSIM", "resolution", 1 );
  mti_PrintFormatted( "[vsim] Resolution = %s\n", entry );
  mti_VsimFree( entry );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Project file

```
[Library]
std = $MODEL_TECH/../std
ieee = $MODEL_TECH/../ieee
verilog = $MODEL_TECH/../verilog
std_developerskit = $MODEL_TECH/../std_developerskit
synopsys = $MODEL_TECH/../synopsys

work = work

[myconfig]
myentry = $MODEL_TECH/xyz

[vcom]
; Turn on VHDL-1993 as the default. Normally is off (VHDL-1987).
; VHDL93 = 1

; Turn on resolving of ambiguous function overloading in favor of the
; "explicit" function declaration (not the one automatically created by
; the compiler for each type declaration). Default is off.
; .ini file has Explict enable so that std_logic_signed/unsigned
; will match synthesis tools behavior.
Explicit = 1

[vlog]

; Turn on converting regular Verilog identifiers to uppercase. Allows case
; insensitivity for module names. Default is no conversion.
; UpCase = 1

; Turns on incremental compilation of modules
; Incremental = 1

[vsim]
; Simulator resolution
; Set to fs, ps, ns, us, ms, or sec with optional prefix of 1, 10, or 100.
Resolution = ns

; User time unit for run commands
; Set to default, fs, ps, ns, us, ms, or sec. The default is to use the
; unit specified for Resolution. For example, if Resolution is 100ps,
; then UserTimeUnit defaults to ps.
UserTimeUnit = default

; Default run length
RunLength = 100

; Maximum iterations that can be run without advancing simulation time
IterationLimit = 5000

; Stop the simulator after an assertion message
; 0 = Note  1 = Warning  2 = Error  3 = Failure  4 = Fatal
BreakOnAssertion = 3

; Default radix for all windows and commands...
; Set to symbolic, ascii, binary, octal, decimal, hex, unsigned
DefaultRadix = symbolic
```

```
; VSIM Startup command
; Startup = do startup.do

; File for saving command transcript
TranscriptFile = transcript

; Specify whether paths in simulator commands should be described
; in VHDL or Verilog format. For VHDL, PathSeparator = /
; for Verilog, PathSeparator = .
PathSeparator = /

; Specify the dataset separator for fully rooted contexts.
; The default is ':'. For example, sim:/top
; Must not be the same character as PathSeparator.
DatasetSeparator = :

; Control VHDL files opened for write
;    0 = Buffered, 1 = Unbuffered
UnbufferedOutput = 0

; Control number of VHDL files open concurrently
;    This number should always be less then the
;    current ulimit setting for max file descriptors
;    0 = unlimited
ConcurrentFileLimit = 40
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# [myconfig] myentry = $MODEL_TECH/xyz
# [myconfig] myentry = .../modeltech/sunos5/xyz
# [Library] std = $MODEL_TECH/../std
# [vsim] Resolution = ns
VSIM 1> run 5
VSIM 2> quit
```

Related Topics

mti_NextRegion()

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FindRegion()

Finds a region by name.

Syntax

region_id = mti_FindRegion(name)

Arguments

Name	Type	Description
name	char *	The name of the region to be found

Return Values

Name	Type	Description
region_id	mtiRegionIdT	A handle to the region or NULL if the region is not found

Description

mti_FindRegion() returns a handle to the specified region. The region name can be either a full hierarchical name or a relative name. A relative name is relative to the current region set by the simulator's environment command. The default current region is the foreign architecture region during elaboration and the top-level region after elaboration is complete.

You can use mti_FindRegion() to obtain a handle to a VHDL, Verilog, or SystemC region. You can use a handle to a Verilog region with PLI functions to obtain information about or access objects in the Verilog region.

During elaboration, mti_FindRegion() will not find design units that have not yet been instantiated.

Examples

FLI code

```
#include <mti.h>

void loadDoneCB( void * param )
{
  char *        region_name;
  mtiRegionIdT regid;

  mti_PrintMessage( "\nLoad Done phase:\n" );

  regid = mti_FindRegion( "top" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  regid = mti_FindRegion( "inst1" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  /* The i1 region is not found here because it is not a subregion
   * of /top, which is the current context.
   */
  regid = mti_FindRegion( "i1" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  regid = mti_FindRegion( "inst1/flip" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  regid = mti_FindRegion( "/top/inst1/toggle" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
```

```
    mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
)
{
  char *       region_name;
  mtiRegionIdT regid;

  mti_AddLoadDoneCB( loadDoneCB, 0 );

  mti_PrintMessage( "\nElaboration phase:\n" );

  regid = mti_FindRegion( "top" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  regid = mti_FindRegion( "inst1" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  regid = mti_FindRegion( "i1" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

  regid = mti_FindRegion( "flip" );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }

 /* The toggle instance is not found here because it has not
  * yet been instantiated.
  */
 regid = mti_FindRegion( "/top/inst1/toggle" );
 if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "Found region %s\n", region_name );
    mti_VsimFree( region_name );
  }
}
```

HDL code

```
    entity for_model is
    end for_model;

    architecture a of for_model is
      attribute foreign of a : architecture is "initForeign for_model.sl";
    begin
    end a;

    entity inv is
      generic ( delay : time := 5 ns );
      port ( a : in bit;
             b : out bit
           );
    end inv;

    architecture b of inv is
    begin
      b <= a after delay;
    end b;

    entity mid is
    end mid;

    architecture a of mid is

      signal s1 : bit := '0';
      signal s2 : bit := '0';
      signal s3 : bit := '0';
      signal s4 : bit := '0';

      component for_model is
      end component;

      for all : for_model use entity work.for_model(a);

      component inv is
        generic ( delay : time := 5 ns );
        port ( a : in bit;
               b : out bit
             );
      end component;

    begin

      flip : inv port map ( s3, s4 );

      i1 : for_model;

      s1 <= not s1 after 5 ns;
      s3 <= not s3 after 5 ns;

      toggle : inv port map ( s1, s2 );

    end a;

    entity top is
```

```
   end top;

   architecture a of top is
     component mid is
     end component;
   begin
     inst1 : mid;
   end a;
```

Simulation output

```
   % vsim -c top
   Reading .../modeltech/tcl/vsim/pref.tcl

   # 5.4b

   # vsim -c top
   # Loading .../modeltech/sunos5/../std.standard
   # Loading work.top(a)
   # Loading work.mid(a)
   # Loading work.inv(b)
   # Loading work.for_model(a)
   # Loading ./for_model.sl
   #
   # Elaboration phase:
   # Found region /top
   # Found region /top/inst1
   # Found region /top/inst1/i1
   # Found region /top/inst1/flip
   #
   # Load Done phase:
   # Found region /top
   # Found region /top/inst1
   # Found region /top/inst1/flip
   # Found region /top/inst1/toggle
   VSIM 1> run 10
   VSIM 2> quit
```

Related Topics

mti_NextRegion()

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FindSignal()

Finds a signal by name.

Syntax

signal_id = mti_FindSignal(name)

Arguments

Name	Type	Description
name	char *	The name of a VHDL or SystemC signal

Return Values

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the VHDL or SystemC signal or NULL if the signal is not found

Description

mti_FindSignal() returns a handle to the specified VHDL or SystemC signal. The signal name can be either a full hierarchical name or a relative name. A relative name is relative to the current region set by the simulator's environment command. The default current region is the foreign architecture region during elaboration and the top-level region after elaboration is complete.

The name of a package signal must include the name of the package.

During elaboration, mti_FindSignal() will not find signals in design units that have not yet been instantiated.

If optimization collapsed the specified name is for a subelement of an input port, the handle that is returned is a handle to the subelement of the actual signal connected to that port.

You cannot use mti_FindSignal() to find slices of arrays.

Examples

FLI code

```
#include "mti.h"

void loadDoneCB( void * param )
{
    char *       region_name;
    mtiSignalIdT sigid;

    mti_PrintMessage( "\nLoad Done phase:\n" );

    sigid = mti_FindSignal( "s1" );
    if ( sigid ) {
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n",
                            region_name, mti_GetSignalName( sigid ) );
        mti_VsimFree( region_name );
    }

    /* Signal p1 is not found here because the current context when
     * elaboration is complete is the top-level design unit and p1
     * exists in the context /top/i1.
     */
    sigid = mti_FindSignal( "p1" );
    if ( sigid ) {
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n",
                            region_name, mti_GetSignalName( sigid ) );
        mti_VsimFree( region_name );
    }

    sigid = mti_FindSignal( "/mypkg/packsig" );
    if ( sigid ) {
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n",
                            region_name, mti_GetSignalName( sigid ) );
        mti_VsimFree( region_name );
    }

    sigid = mti_FindSignal( "/top/s2" );
    if ( sigid ) {
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n",
                            region_name, mti_GetSignalName( sigid ) );
        mti_VsimFree( region_name );
    }

    sigid = mti_FindSignal( "/top/s3(0)" );
    if ( sigid ) {
        char * signal_name = mti_GetSignalNameIndirect( sigid, 0, 0 );
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n", region_name, signal_name );
        mti_VsimFree( region_name );
        mti_VsimFree( signal_name );
    }
```

```
        sigid = mti_FindSignal( "toggle/a" );
        if ( sigid ) {
            region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
            mti_PrintFormatted( "Found signal %s/%s\n",
                                region_name, mti_GetSignalName( sigid ) );
            mti_VsimFree( region_name );
        }
    }

    void initForeign(
        mtiRegionIdT     region,   /* The ID of the region in which this     */
                                   /* foreign architecture is instantiated.  */
        char            *param,    /* The last part of the string in the     */
                                   /* foreign attribute.                     */
        mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
        mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
    )
    {
        char *        region_name;
        mtiSignalIdT sigid;

        mti_AddLoadDoneCB( loadDoneCB, 0 );

        mti_PrintMessage( "\nElaboration phase:\n" );

        /* Signal s1 is not found here because the current context during
         * elaboration is the context of the foreign architecture and s1
         * exists in the context /top.
         */
        sigid = mti_FindSignal( "s1" );
        if ( sigid ) {
            region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
            mti_PrintFormatted( "Found signal %s/%s\n",
                                region_name, mti_GetSignalName( sigid ) );
            mti_VsimFree( region_name );
        }

        sigid = mti_FindSignal( "p1" );
        if ( sigid ) {
            region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
            mti_PrintFormatted( "Found signal %s/%s\n",
                                region_name, mti_GetSignalName( sigid ) );
            mti_VsimFree( region_name );
        }

        sigid = mti_FindSignal( "/mypkg/packsig" );
        if ( sigid ) {
            region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
            mti_PrintFormatted( "Found signal %s/%s\n",
                                region_name, mti_GetSignalName( sigid ) );
            mti_VsimFree( region_name );
        }

        sigid = mti_FindSignal( "/top/s2" );
        if ( sigid ) {
            region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
            mti_PrintFormatted( "Found signal %s/%s\n",
                                region_name, mti_GetSignalName( sigid ) );
```

```
        mti_VsimFree( region_name );
    }

    sigid = mti_FindSignal( "/top/s3(4)" );
    if ( sigid ) {
        char * signal_name = mti_GetSignalNameIndirect( sigid, 0, 0 );
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n", region_name, signal_name );
        mti_VsimFree( region_name );
        mti_VsimFree( signal_name );
    }

    /* Signal /top/toggle/a is not found because the toggle instance has
     * not yet been elaborated.
     */
    sigid = mti_FindSignal( "/top/toggle/a" );
    if ( sigid ) {
        region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ) );
        mti_PrintFormatted( "Found signal %s/%s\n",
                            region_name, mti_GetSignalName( sigid ) );
        mti_VsimFree( region_name );
    }
}
```

HDL code

```
package mypkg is
  signal packsig : bit := '0';
end mypkg;

entity for_model is
  port ( p1 : in bit );
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

use work.mypkg.all;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit_vector( 7 downto 0 ) := "01101010";

  component for_model is
    port ( p1 : in bit );
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  i1 : for_model port map ( s1 );

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;
  packsig <= not packsig after 5 ns;
```

```
    toggle : inv port map ( s1, s2 );

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.5

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.mypkg
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
# Found signal /top/i1/p1
# Found signal /mypkg/packsig
# Found signal /top/s2
# Found signal /top/s3(4)
# Loading work.inv(b)
#
# Load Done phase:
# Found signal /top/s1
# Found signal /mypkg/packsig
# Found signal /top/s2
# Found signal /top/s3(0)
# Found signal /top/toggle/a
VSIM 1> run 10
VSIM 2> quit
```

Related Topics

mti_NextRegion()

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FindVar()

Finds a VHDL variable, generic, or constant by name.

Syntax

variable_id = mti_FindVar(name)

Arguments

Name	Type	Description
name	char *	The name of a VHDL variable, generic, or constant

Return Values

Name	Type	Description
variable_id	mtiVariableIdT	A handle to the VHDL variable, generic, or constant or NULL if the object is not found

Description

mti_FindVar() returns a handle to the specified VHDL variable, generic, or constant. The name can be either a full hierarchical name or a relative name. A relative name is relative to the current region set by the simulator's environment command. The default current region is the top-level region. For objects declared in a process, the name must include the process label.

You can:

- call mti_FindVar() successfully only after elaboration is complete.

- cannot use mti_FindVar() to find slices of arrays.

- cannot use mti_FindVar() to find a process variable when mti_FindVar() is called from a foreign subprogram that is called from the process where the variable is declared.

Examples

FLI code

```
#include "mti.h"

static void printVarInfo( mtiVariableIdT varid )
{
    if ( varid ) {
        mti_PrintFormatted( "Found variable %s\n", mti_GetVarName( varid ) );
    }
}

void loadDoneCB( void * param )
{
    mti_PrintMessage( "\nLoad Done phase:\n" );

    printVarInfo( mti_FindVar( "/TOP/p1/v1" ) );
    printVarInfo( mti_FindVar( "/p1/const1" ) );
    printVarInfo( mti_FindVar( "c1" ) );
    printVarInfo( mti_FindVar( "/top/sv1" ) );
    printVarInfo( mti_FindVar( "/top/TOGGLE/proc1/count" ) );
    printVarInfo( mti_FindVar( "/toggle/delay" ) );
    printVarInfo( mti_FindVar( "/top/toggle/myconst" ) );
    printVarInfo( mti_FindVar( "/my_pkg/psv1" ) );
}

void initForeign(
    mtiRegionIdT      region,   /* The ID of the region in which this      */
                                /* foreign architecture is instantiated.   */
    char             *param,    /* The last part of the string in the      */
                                /* foreign attribute.                      */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
    mti_AddLoadDoneCB( loadDoneCB, 0 );

    mti_PrintMessage( "\nElaboration phase:\n" );

    printVarInfo( mti_FindVar( "/top/p1/v1" ) );
    printVarInfo( mti_FindVar( "/top/p1/const1" ) );
    printVarInfo( mti_FindVar( "/top/c1" ) );
    printVarInfo( mti_FindVar( "/top/sv1" ) );
    printVarInfo( mti_FindVar( "/top/toggle/proc1/count" ) );
    printVarInfo( mti_FindVar( "/top/toggle/delay" ) );
    printVarInfo( mti_FindVar( "/top/toggle/myconst" ) );
}
```

HDL code

```
package my_pkg is
   shared variable psv1 : bit := '1';
end my_pkg;

entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
   constant myconst : real := 13.78;
begin
  b <= a after delay;

  proc1 : process
    variable count : integer := 0;
  begin
    count := count + 1;
    wait on a;
  end process;
end b;

use work.my_pkg.all;

entity top is
end top;

architecture a of top is

  constant c1 : integer := 42;
  shared variable sv1 : integer := 0;

  signal s1 : bit := '0';
  signal s2 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;
```

```
begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

  toggle : inv port map ( s1, s2 );

  p1 : process
    constant const1 : integer := 4;
    variable v1     : integer := 0;
  begin
    v1 := v1 + const1;
    sv1 := sv1 + 1;
    psv1 := not psv1;
    wait for 5 ns;
  end process;

  p2 : process
  begin
    sv1 := sv1 + 1;
    wait for 3 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.my_pkg
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
# Loading work.inv(b)
#
# Load Done phase:
# Found variable v1
# Found variable const1
# Found variable c1
# Found variable sv1
# Found variable count
# Found variable delay
# Found variable myconst
# Found variable psv1
VSIM 1> run 10
VSIM 2> quit
```

Related Topics

mti_NextRegion()

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FirstLowerRegion()

Gets the first subregion inside of a region.

Syntax

subregion_id = mti_FirstLowerRegion(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to the region from which the first subregion is to be obtained

Return Values

Name	Type	Description
subregion_id	mtiRegionIdT	A handle to the first subregion inside a region or NULL if there are no subregions

Description

mti_FirstLowerRegion() returns a handle to the first subregion of the specified region. Use mti_NextRegion() to get the subsequent subregions of the specified region.

mti_FirstLowerRegion() will return a handle to a VHDL, Verilog, or SystemC region. Verilog regions include tasks and functions. You can use a handle to a Verilog region with PLI functions to obtain information about or access objects in the Verilog region.

During elaboration, mti_FirstLowerRegion() wil not find design units that have not yet been instantiated.

Examples

FLI code

```c
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
      printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT      region,    /* The ID of the region in which this
    */
                               /* foreign architecture is
instantiated.  */
  char             *param,    /* The last part of the string in
the      */
                               /* foreign
attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign
model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign
model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
```

```
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#  Region /top
#   Region /top/inst1
#     Region /top/inst1/i1
#     Region /top/inst1/flip
#
# Load Done phase:
#  Region /top
#   Region /top/inst1
#     Region /top/inst1/flip
#     Region /top/inst1/i1
#     Region /top/inst1/toggle
VSIM 1> run 20
VSIM 2> quit
```

Related Topics

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FirstProcess()

Gets the first process in a region.

Syntax

process_id = mti_FirstProcess(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL or SystemC region

Return Values

Name	Type	Description
process_id	mtiProcessIdT	A handle to the first VHDL or SystemC process in a region or NULL if there are no processes in the region

Description

mti_FirstProcess() returns a handle to the first process in the specified region. You can use mti_NextProcess() to get the subsequent processes in the specified region.

mti_FirstProcess() resets the region used by previous calls to mti_FirstProcess() and mti_NextProcess(); therefore, mti_NextProcess() always uses the region set by the latest call to mti_FirstProcess().

Examples

FLI code

```
#include <mti.h>

void printProcesses( mtiRegionIdT region, int indent )
{
  mtiProcessIdT procid;

  for ( procid = mti_FirstProcess( region ); procid;
        procid = mti_NextProcess() ) {
      if ( procid ) {
          mti_PrintFormatted( "%*cProcess %s\n", indent, ' ',
                              mti_GetProcessName( procid ) );
      }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printProcesses( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
      printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
  char               *param,    /* The last part of the string in the    */
                                /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;

  p1 : process
    variable count : integer := 0;
  begin
    count := count + 1;
    wait on a;
  end process;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
```

```
   s3 <= not s3 after 5 ns;

   toggle : inv port map ( s1, s2 );

end a;

entity top is
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#  Region /top
#   Region /top/inst1
#    Region /top/inst1/i1
#    Region /top/inst1/flip
#     Process p1
#     Process line__19
#
# Load Done phase:
#  Region /top
#   Region /top/inst1
#    Process line__58
#    Process line__57
#    Region /top/inst1/flip
#     Process p1
#     Process line__19
#    Region /top/inst1/i1
#    Region /top/inst1/toggle
#     Process p1
#     Process line__19
VSIM 1> run 20
VSIM 2> quit
```

Related Topics

mti_GetSignalSubelements()

---

mti_VsimFree()

mti_TickLength()

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mti_FirstSignal()

Gets the first signal in a region.

Syntax

signal_id = mti_FirstSignal(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL or SystemC region

Return Values

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the first VHDL or SystemC signal in a region or NULL if there are no signals in the region

Description

mti_FirstSignal() returns a handle to the first VHDL or SystemC signal in the specified region. You can use mti_NextSignal() to get the subsequent VHDL or SystemC signals in the specified region.

mti_FirstSignal() resets the region used by previous calls to mti_FirstSignal() and mti_NextSignal(); therefore, mti_NextSignal() always uses the region set by the latest call to mti_FirstSignal().

Examples

FLI code

```c
#include <mti.h>

void printSignals( mtiRegionIdT region, int indent )
{
  mtiSignalIdT sigid;

  for ( sigid = mti_FirstSignal( region ); sigid;
        sigid = mti_NextSignal() ) {
     if ( sigid ) {
        mti_PrintFormatted( "%*cSignal %s\n", indent, ' ',
                            mti_GetSignalName( sigid ) );
     }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printSignals( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
     printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
   signal count : integer := 0;
begin
  b <= a after delay;

  p1 : process( a )
  begin
    count <= count + 1 after 0 ns;
  end process;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;
```

```
      toggle : inv port map ( s1, s2 );

   end a;

   entity top is
   end top;

   architecture a of top is
     component mid is
     end component;
   begin
     inst1 : mid;
   end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#  Region /top
#   Region /top/inst1
#     Signal s4
#     Signal s3
#     Signal s2
#     Signal s1
#     Region /top/inst1/i1
#     Region /top/inst1/flip
#      Signal count
#      Signal b
#      Signal a
#
# Load Done phase:
#  Region /top
#   Region /top/inst1
#     Signal s1
#     Signal s2
#     Signal s3
#     Signal s4
#     Region /top/inst1/flip
#      Signal a
#      Signal b
#      Signal count
#     Region /top/inst1/i1
#     Region /top/inst1/toggle
#      Signal a
#      Signal b
#      Signal count
VSIM 1> run 10
VSIM 2> quit
```

Related Topics

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FirstVar()

Gets the first VHDL variable, generic, or constant visible to a process.

Syntax

variable_id = mti_FirstVar(process_id)

Arguments

Name	Type	Description
process_id	mtiProcessIdT	A handle to a VHDL process

Return Values

Name	Type	Description
variable_id	mtiVariableIdT	A handle to the first VHDL variable, generic, or constant visible to a process or NULL if none of these objects are visible to the process

Description

mti_FirstVar() returns a handle to the first VHDL variable, generic, or constant visible to the specified process. You can use mti_NextVar() to get the subsequent VHDL variables, generics, and constants visible to the specified process.

All generics of an entity are visible to every process within the associated architecture.

mti_FirstVar() resets the process used by previous calls to mti_FirstVar() and mti_NextVar(); therefore, mti_NextVar() always uses the process set by the latest call to mti_FirstVar().

Examples

FLI code

```c
#include <mti.h>

void printVariables( mtiProcessIdT process, int indent )
{
  mtiVariableIdT varid;

  for ( varid = mti_FirstVar( process ); varid; varid = mti_NextVar() ) {
    if ( varid ) {
      mti_PrintFormatted( "%*cVariable %s\n", indent, ' ',
                          mti_GetVarName( varid ) );
    }
  }
}

void printProcesses( mtiRegionIdT region, int indent )
{
  mtiProcessIdT procid;

  for ( procid = mti_FirstProcess( region ); procid;
        procid = mti_NextProcess() ) {
    if ( procid ) {
      mti_PrintFormatted( "%*cProcess %s\n", indent, ' ',
                          mti_GetProcessName( procid ) );
      printVariables( procid, indent+2 );
    }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printProcesses( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the    */
```

```
                                 /* foreign attribute.                      */
   mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
   mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
   mti_AddLoadDoneCB( loadDoneCB, 0 );
   mti_PrintMessage( "\nElaboration phase:\n" );
   printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;

  p1 : process
    constant increment : integer := 1;
    variable count : integer := 0;
  begin
    count := count + increment;
    wait on a;
  end process;
end b;

entity mid is
  generic ( gen1 : string := "Mid" );
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  testproc : process
    constant c1 : string := "mystring";
    variable v1 : bit := '0';
```

```
      variable v2 : integer := 42;
      variable v3 : real := 7.82;
    begin
      v1 := not v1;
      v2 := v2 + 2;
      v3 := v3 + 1.5;
      wait for 5 ns;
    end process;

    flip : inv port map ( s3, s4 );

    i1 : for_model;

    s1 <= not s1 after 5 ns;
    s3 <= not s3 after 5 ns;

    toggle : inv port map ( s1, s2 );

  end a;

  entity top is
  end top;

  architecture a of top is
    component mid is
      generic ( gen1 : string := "Top" );
    end component;
  begin
    inst1 : mid;
  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#  Region /top
#   Region /top/inst1
#    Process testproc
#     Variable gen1
#     Variable c1
#     Variable v1
#     Variable v2
#     Variable v3
#    Region /top/inst1/i1
#    Region /top/inst1/flip
#     Process p1
#      Variable delay
#      Variable increment
#      Variable count
#    Process line__19
#      Variable delay
#
# Load Done phase:
#  Region /top
#   Region /top/inst1
#    Process line__72
#     Variable gen1
#    Process line__71
#     Variable gen1
#    Process testproc
#     Variable gen1
#     Variable c1
#     Variable v1
#     Variable v2
#     Variable v3
#    Region /top/inst1/flip
#     Process p1
#      Variable delay
#      Variable increment
#      Variable count
#    Process line__19
#      Variable delay
#    Region /top/inst1/i1
#    Region /top/inst1/toggle
#     Process p1
#      Variable delay
#      Variable increment
```

```
#        Variable count
#      Process line__19
#       Variable delay
VSIM 1> run 10
VSIM 2> quit
```

Related Topics

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_FirstVarByRegion()

Gets the first SystemC variable or constant visible in a region.

Syntax

variable_id = mti_FirstVarByRegion(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
variable_id	mtiVariableIdT	A handle to the first SystemC variable or constant visible to the region or NULL if none of these objects are visible to the region.

Description

mti_FirstVarByRegion() returns a handle to the first SystemC variable or constant visible to the specified region in much that same way as mti_FirstVar() returns a handle to the first VHDL variable, generic, or constant visible to the specified process. You can use mti_NextVar() to get the subsequent SystemC variables and constants visible to the specified region.

Both mti_FirstVarByRegion() and mti_FirstVar() reset the process/region used by mti_NextVar(); therefore, mti_NextVar() always uses the process or region set by the latest call to mti_FirstVarByRegion() or mti_FirstVar().

Variables are not visible in VHDL or Verilog regions but are visible in VHDL or Verilog processes. Conversely, variables ARE visible in SystemC regions, but are NOT visible in SystemC processes. So this function is needed in order to get SystemC variables (which cannot be accessed with mti_FirstVar()).

Related Topics

mti_GetSignalSubelements()

mti_VsimFree()

mti_TickLength()

mti_ForceSignal()

Forces a value onto a VHDL signal.

Syntax

error_code = mti_ForceSignal(signal_id, value_string, delay, force_type, cancel_period, repeat_period)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the VHDL signal to be forced
value_string	char *	The value to be forced specified as a string in the same format as would be provided to the simulator's force command
delay	mtiDelayT	The time at which the force is to be applied relative to the current time; specified in current simulator resolution units
force_type	mtiForceTypeT	Indicates whether the force is to freeze, drive, deposit, or use the default force type
cancel_period	mtiInt32T	If non-negative, specifies the period after which the force is canceled; specified in current simulator resolution units
repeat_period	mtiInt32T	If non-negative, specifies the period in which the force is repeated; specified in current simulator resolution units

Return Values

Name	Type	Description
error_code	int	1 if successful; 0 if there is an error

Description

mti_ForceSignal() forces the specified VHDL signal to the specified value using the specified force type and an optional delay, cancel period, and repeat period. The value must be specified in a string in the same format as would be provided to the simulator's force command, and the

restrictions on the type of the value are the same as for the force command (refer to the Command Reference Manual for details).

If the delay parameter is non-negative, then the delay specifies the time at which the force is to be applied relative to the current time. If the delay parameter is negative, then the force is applied immediately.

If the cancel_period parameter is non-negative, then the force is canceled after the specified period. If the cancel_period parameter is negative, then the force is not automatically canceled.

If the repeat_period parameter is non-negative, then the force is repeated for the specified period. If the repeat_period parameter is negative, then the force is not automatically repeated.

To force records or arrays that are not one-dimensional arrays of character enumerations, use mti_GetSignalSubelements() to get a handle to each element and force each element individually.

mti_ForceSignal() cannot force a port if the port has values coming into it from a higher level or if the port has a conversion function on it (although in some cases you might be able to force the port using the MTI_FORCE_DRIVE force type with mti_ForceSignal()).

Examples

FLI code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef WIN32
#include <unistd.h>
#endif

#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  struct signalInfoT_tag * child;
  char                   * name;
  void                   * last_value;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
  mtiTypeKindT             typekind;
  mtiDirectionT            direction;
  char                     granulate;
} signalInfoT;

typedef struct {
  signalInfoT    * sig_info;        /* List of signals.         */
  mtiProcessIdT    proc;            /* Test process id.         */
  int              state;           /* Current state of test.   */
} instanceInfoT;

static void forceSignal(
  mtiSignalIdT sigid,
  mtiTypeIdT   sigtypeid,
  mtiTypeKindT sigtypekind,
  int          state
)
{
  int            i;
  int            result = 1;
  mtiSignalIdT *elem_list;
  mtiSignalIdT  elem_sigid;
  mtiTypeIdT    elem_typeid;

  switch ( sigtypekind ) {
   case MTI_TYPE_SCALAR:
    switch ( state ) {
      case 0:
       result = mti_ForceSignal(sigid, "42", -1, MTI_FORCE_FREEZE, -1, -1);
       break;
      case 2:
       result = mti_ForceSignal(sigid, "120", 1, MTI_FORCE_FREEZE, 7, -1);
       break;
      case 4:
       result = mti_ForceSignal(sigid, "777", -1, MTI_FORCE_DEPOSIT, -1, 2);
       break;
      }
```

```
     break;
   case MTI_TYPE_ARRAY:
    elem_typeid = mti_GetArrayElementType( sigtypeid );
    if ( mti_GetTypeKind( elem_typeid ) == MTI_TYPE_ENUM ) {
      /* NOTE: ASSUMING ARRAY OF LENGTH 4 ! */
     if ( mti_TickLength( elem_typeid ) == 9 ) {  /* ASSUME std_logic */
      switch ( state ) {
       case 0:
        result = mti_ForceSignal( sigid, "ZW1H", -1,
                                  MTI_FORCE_FREEZE, -1, -1 );
        break;
       case 2:
        result = mti_ForceSignal( sigid, "LLLL", 1,
                                  MTI_FORCE_FREEZE, 7, -1 );
        break;
       case 4:
        result = mti_ForceSignal( sigid, "1-1-", -1,
                                  MTI_FORCE_DEPOSIT, -1, 2 );
        break;
      }
     } else {  /* ASSUME bit */
      switch ( state ) {
       case 0:
        result = mti_ForceSignal( sigid, "0011", -1,
                                  MTI_FORCE_FREEZE, -1, -1 );
        break;
       case 2:
        result = mti_ForceSignal( sigid, "1000", 1,
                                  MTI_FORCE_FREEZE, 7, -1 );
        break;
       case 4:
        result = mti_ForceSignal( sigid, "0010", -1,
                                  MTI_FORCE_DEPOSIT, -1, 2 );
        break;
      }
     }
    } else {
     elem_list = mti_GetSignalSubelements( sigid, 0 );
     for ( i = 0; i < mti_TickLength( sigtypeid ); i++ ) {
      elem_sigid  = elem_list[i];
      elem_typeid = mti_GetSignalType( elem_sigid );
      forceSignal( elem_sigid, elem_typeid,
                   mti_GetTypeKind( elem_typeid ), state );
     }
     mti_VsimFree( elem_list );
    }
    break;
   case MTI_TYPE_RECORD:
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    for ( i = 0; i < mti_GetNumRecordElements( sigtypeid ); i++ ) {
     elem_sigid  = elem_list[i];
     elem_typeid = mti_GetSignalType( elem_sigid );
     forceSignal( elem_sigid, elem_typeid,
                  mti_GetTypeKind( elem_typeid ), state );
    }
    mti_VsimFree( elem_list );
    break;
   case MTI_TYPE_ENUM:
```

```
      if ( mti_TickLength( sigtypeid ) == 9 ) {   /* ASSUME std_logic */
       switch ( state ) {
        case 0:
         result = mti_ForceSignal( sigid, "'W'", -1,
                                   MTI_FORCE_FREEZE, -1, -1 );
         break;
        case 2:
         result = mti_ForceSignal( sigid, "'0'", 1,
                                   MTI_FORCE_FREEZE, 7, -1 );
         break;
        case 4:
         result = mti_ForceSignal( sigid, "'H'", -1,
                                   MTI_FORCE_DEPOSIT, -1, 2 );
         break;
       }
      } else {
       switch ( state ) {   /* ASSUME bit */
        case 0:
         result = mti_ForceSignal( sigid, "0", -1,
                                   MTI_FORCE_FREEZE, -1, -1 );
         break;
        case 2:
         result = mti_ForceSignal( sigid, "1", 1,
                                   MTI_FORCE_FREEZE, 7, -1 );
         break;
        case 4:
         result = mti_ForceSignal( sigid, "0", -1,
                                   MTI_FORCE_DEPOSIT, -1, 2 );
         break;
       }
      }
      break;
     default:
      break;
    }
    if ( ! result ) {
     fprintf( stderr, "Error in signal force.\n" );
    }
}

static void releaseSignal(
  mtiSignalIdT sigid,
  mtiTypeIdT   sigtypeid,
  mtiTypeKindT sigtypekind
)
{
   int          i;
   mtiSignalIdT *elem_list;
   mtiSignalIdT  elem_sigid;
   mtiTypeIdT    elem_typeid;

   switch ( sigtypekind ) {
    case MTI_TYPE_SCALAR:
    case MTI_TYPE_ENUM:
    case MTI_TYPE_TIME:
     if ( ! mti_ReleaseSignal( sigid ) ) {
      fprintf( stderr, "Error in signal release.\n" );
     }
```

```
     break;
    case MTI_TYPE_ARRAY:
     elem_typeid = mti_GetArrayElementType( sigtypeid );
     if ( mti_GetTypeKind( elem_typeid ) == MTI_TYPE_ENUM ) {
      if ( ! mti_ReleaseSignal( sigid ) ) {
       fprintf( stderr, "Error in signal release.\n" );
      }
     } else {
      elem_list = mti_GetSignalSubelements( sigid, 0 );
      for ( i = 0; i < mti_TickLength( sigtypeid ); i++ ) {
       elem_sigid  = elem_list[i];
       elem_typeid = mti_GetSignalType( elem_sigid );
       releaseSignal( elem_sigid, elem_typeid,
                    mti_GetTypeKind( elem_typeid ) );
      }
      mti_VsimFree( elem_list );
     }
     break;
    case MTI_TYPE_RECORD:
     elem_list = mti_GetSignalSubelements( sigid, 0 );
     for ( i = 0; i < mti_GetNumRecordElements( sigtypeid ); i++ ) {
      elem_sigid  = elem_list[i];
      elem_typeid = mti_GetSignalType( elem_sigid );
      releaseSignal( elem_sigid, elem_typeid,
                    mti_GetTypeKind( elem_typeid ) );
     }
     mti_VsimFree( elem_list );
     break;
    default:
     break;
   }
}

static void testForce( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  signalInfoT   *siginfo;

  switch ( inst_data->state ) {
   case 0:
   case 2:
   case 4:
    for (siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next) {
     forceSignal( siginfo->sigid, siginfo->typeid,
                siginfo->typekind, inst_data->state );
    }
    break;
   case 1:
   case 3:
   case 5:
    for (siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next) {
     releaseSignal( siginfo->sigid, siginfo->typeid, siginfo->typekind );
    }
    break;
   default:
    break;
   }
```

```
      inst_data->state++;
      mti_ScheduleWakeup( inst_data->proc, 10 );
    }

  static signalInfoT * setupSignal( mtiSignalIdT sigid )
  {
    signalInfoT * siginfo;

    siginfo = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
    siginfo->sigid       = sigid;
    siginfo->name        = mti_GetSignalNameIndirect( sigid, 0, 0 );
    siginfo->typeid      = mti_GetSignalType( sigid );
    siginfo->typekind    = mti_GetTypeKind( siginfo->typeid );
    siginfo->direction   = mti_GetSignalMode( sigid );
    siginfo->last_value  = mti_GetSignalValueIndirect( sigid, 0 );
    siginfo->child       = 0;
    siginfo->next        = 0;

    /* For records and arrays of composites, we want to set/drive
     * values at the subelement level.  For scalars and arrays of
     * scalars, we want to set/drive values at the top level.
     */
    switch ( siginfo->typekind ) {
     case MTI_TYPE_ARRAY:
      switch( mti_GetTypeKind(mti_GetArrayElementType(siginfo->typeid)) ) {
       case MTI_TYPE_ARRAY:
       case MTI_TYPE_RECORD:
        siginfo->granulate = 1;
        break;
       default:
        siginfo->granulate = 0;
        break;
      }
      break;
     case MTI_TYPE_RECORD:
      siginfo->granulate = 1;
      break;
     default:
      siginfo->granulate = 0;
      break;
    }

    if ( siginfo->granulate ) {
      signalInfoT  * eleminfo;
      signalInfoT  * currinfo;
      int            i;
      mtiSignalIdT * subelem;

      subelem = mti_GetSignalSubelements( siginfo->sigid, 0 );
      for ( i = 0; i < mti_TickLength(siginfo->typeid); i++ ) {
       eleminfo = setupSignal( subelem[i] );
       if ( siginfo->child == 0 ) {
        siginfo->child = eleminfo;
       } else {
        currinfo->next = eleminfo;
       }
       currinfo = eleminfo;
      }
```

```
        mti_VsimFree( subelem );
    }

    return( siginfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiRegionIdT    region;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;
  inst_data->state    = 0;
  region              = mti_GetTopRegion();

  for (sigid = mti_FirstSignal( region ); sigid; sigid = mti_NextSignal()) {
   siginfo = setupSignal( sigid );
   if ( inst_data->sig_info == 0 ) {
    inst_data->sig_info = siginfo;
   }
   else {
    curr_info->next = siginfo;
   }
    curr_info = siginfo;
  }

  inst_data->proc = mti_CreateProcess( "Test Process", testForce,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 11 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

package typepkg is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )      of integer;

  type rectype is record
    a : bit;
    b : integer;
    c : std_logic;
  end record;

end package typepkg;

-- -- --

entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

-- -- --

library ieee;
use ieee.std_logic_1164.all;

use work.typepkg.all;

entity top is
end top;

architecture a of top is

  signal bitsig1      : bit        := '1';
  signal intsig1      : integer    := 21;
  signal stdlogicsig1 : std_logic := 'H';

  signal bitarr1      : bitarray  := "0110";
  signal stdlogicarr1 : std_logic_vector( 1 to 4 ) := "-X0U";
  signal intarr1      : intarray  := ( 10, 11, 12 );

  signal rec1         : rectype   := ( '0', 1, 'X' );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
    bitsig1       <= not bitsig1 after 5 ns;
    intsig1       <= intsig1 + 1 after 5 ns;
    stdlogicsig1 <= '-' after 5 ns when stdlogicsig1 = 'H' else
                    'U' after 5 ns when stdlogicsig1 = '-' else
                    'X' after 5 ns when stdlogicsig1 = 'U' else
                    '0' after 5 ns when stdlogicsig1 = 'X' else
                    '1' after 5 ns when stdlogicsig1 = '0' else
                    'Z' after 5 ns when stdlogicsig1 = '1' else
                    'W' after 5 ns when stdlogicsig1 = 'Z' else
                    'L' after 5 ns when stdlogicsig1 = 'W' else
                    'H' after 5 ns;

bitarr1              <= not bitarr1 after 5 ns;

intarr1(1)           <= intarr1(1) + 1 after 5 ns;
intarr1(2)           <= intarr1(2) + 1 after 5 ns;
intarr1(3)           <= intarr1(3) + 1 after 5 ns;

stdlogicarr1(1) <= '-' after 5 ns when stdlogicarr1(1) = 'H' else
                    'U' after 5 ns when stdlogicarr1(1) = '-' else
                    'X' after 5 ns when stdlogicarr1(1) = 'U' else
                    '0' after 5 ns when stdlogicarr1(1) = 'X' else
                    '1' after 5 ns when stdlogicarr1(1) = '0' else
                    'Z' after 5 ns when stdlogicarr1(1) = '1' else
                    'W' after 5 ns when stdlogicarr1(1) = 'Z' else
                    'L' after 5 ns when stdlogicarr1(1) = 'W' else
                    'H' after 5 ns;

stdlogicarr1(2) <= '-' after 5 ns when stdlogicarr1(2) = 'H' else
                    'U' after 5 ns when stdlogicarr1(2) = '-' else
                    'X' after 5 ns when stdlogicarr1(2) = 'U' else
                    '0' after 5 ns when stdlogicarr1(2) = 'X' else
                    '1' after 5 ns when stdlogicarr1(2) = '0' else
                    'Z' after 5 ns when stdlogicarr1(2) = '1' else
                    'W' after 5 ns when stdlogicarr1(2) = 'Z' else
                    'L' after 5 ns when stdlogicarr1(2) = 'W' else
                    'H' after 5 ns;

stdlogicarr1(3) <= '-' after 5 ns when stdlogicarr1(3) = 'H' else
                    'U' after 5 ns when stdlogicarr1(3) = '-' else
                    'X' after 5 ns when stdlogicarr1(3) = 'U' else
                    '0' after 5 ns when stdlogicarr1(3) = 'X' else
                    '1' after 5 ns when stdlogicarr1(3) = '0' else
                    'Z' after 5 ns when stdlogicarr1(3) = '1' else
                    'W' after 5 ns when stdlogicarr1(3) = 'Z' else
                    'L' after 5 ns when stdlogicarr1(3) = 'W' else
                    'H' after 5 ns;

stdlogicarr1(4) <= '-' after 5 ns when stdlogicarr1(4) = 'H' else
                    'U' after 5 ns when stdlogicarr1(4) = '-' else
                    'X' after 5 ns when stdlogicarr1(4) = 'U' else
                    '0' after 5 ns when stdlogicarr1(4) = 'X' else
                    '1' after 5 ns when stdlogicarr1(4) = '0' else
                    'Z' after 5 ns when stdlogicarr1(4) = '1' else
                    'W' after 5 ns when stdlogicarr1(4) = 'Z' else
                    'L' after 5 ns when stdlogicarr1(4) = 'W' else
                    'H' after 5 ns;
```

```
rec1.a <= not rec1.a after 5 ns;
rec1.b <= rec1.b + 1 after 5 ns;
rec1.c <= '-' after 5 ns when rec1.c = 'H' else
          'U' after 5 ns when rec1.c = '-' else
          'X' after 5 ns when rec1.c = 'U' else
          '0' after 5 ns when rec1.c = 'X' else
          '1' after 5 ns when rec1.c = '0' else
          'Z' after 5 ns when rec1.c = '1' else
          'W' after 5 ns when rec1.c = 'Z' else
          'L' after 5 ns when rec1.c = 'W' else
          'H' after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.typepkg
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> add list -w 1 /top/bitsig1
VSIM 2> add list -w 3 /top/intsig1
VSIM 3> add list -w 1 /top/stdlogicsig1
VSIM 4> add list -w 4 /top/bitarr1
VSIM 5> add list -w 4 /top/stdlogicarr1
VSIM 6> add list -w 15 /top/intarr1
VSIM 7> add list -w 10 /top/rec1
VSIM 8> run 70
VSIM 9> write list list.out
VSIM 10> quit -f
% cat list.out
 ns       /top/bitsig1                     /top/intarr1  /top/rec1
  delta        /top/intsig1
           /top/stdlogicsig1
                 /top/bitarr1
                 /top/stdlogicarr1
  0  +0         1  21 H 0110 -X0U    {10 11 12}   {0 1 X}
  5  +0         0  22 - 1001 U01X    {11 12 13}   {1 2 0}
 10  +0         1  23 U 0110 X1Z0    {12 13 14}   {0 3 1}
 11  +0         0  42 W 0011 ZW1H    {42 42 42}   {0 42 W}
 21  +1         1  43 L 1100 WLZ-    {43 43 43}   {1 43 L}
 26  +0         0  44 H 0011 LHWU    {44 44 44}   {0 44 H}
 31  +0         1  45 - 1100 H-LX    {45 45 45}   {1 45 -}
 32  +0         1 120 0 1000 LLLL    {120 120 120} {1 120 0}
 38  +1         0 121 1 0111 HHHH    {121 121 121} {0 121 1}
 43  +0         1 122 Z 1000 ----    {122 122 122} {1 122 Z}
 48  +0         0 123 W 0111 UUUU    {123 123 123} {0 123 W}
 51  +0         0 777 H 0010 1-1-    {777 777 777} {0 777 H}
 53  +0         0 777 H 0010 1-1-    {777 777 777} {0 777 H}
 56  +0         0 778 - 1101 ZUZU    {778 778 778} {0 778 -}
 57  +0         0 777 H 0010 1-1-    {777 777 777} {0 777 H}
 58  +0         1 777 H 0010 1-1-    {777 777 777} {1 777 H}
 59  +0         0 777 H 0010 1-1-    {777 777 777} {0 777 H}
 61  +1         1 778 - 1101 ZUZU    {778 778 778} {1 778 -}
 66  +0         0 779 U 0010 WXWX    {779 779 779} {0 779 U}
```

Related Topics

mti_VsimFree()

mti_TickLength()

mti_Free()

Frees simulator-managed memory.

Syntax

mti_Free(pointer)

Arguments

Name	Type	Description
pointer	void *	A pointer to some memory previously allocated by mti_Malloc()

Return Values

Nothing

Description

mti_Free() returns the specified block of memory allocated by mti_Malloc() to the vsim memory allocator. You cannot use mti_Free() for memory allocated by direct calls to malloc().

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  mtiSignalIdT  sigid;
  mtiProcessIdT procid;
} instanceInfoT;

char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}

void watchSignal( void * param )
{
  char           * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T        sigval;

  region_name = mti_GetRegionFullName( mti_GetSignalRegion(inst->sigid) );
  sigval      = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      region_name, mti_GetSignalName( inst->sigid ),
                      convertStdLogicValue( sigval ) );
  mti_VsimFree( region_name );

  if ( mti_Now() >= 30 ) {
```

```
      mti_PrintMessage( "Turning off signal watcher.\n" );
      mti_Free( inst );
    } else {
      mti_ScheduleWakeup( inst->procid, 5 );
    }
  }
  void initForeign(
    mtiRegionIdT        region,    /* The ID of the region in which this     */
                                   /* foreign architecture is instantiated.  */
    char               *param,     /* The last part of the string in the     */
                                   /* foreign attribute.                     */
    mtiInterfaceListT *generics,   /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports       /* A list of ports for the foreign model.  */
  )
  {
    instanceInfoT * inst;

    inst        = (instanceInfoT *) mti_Malloc( sizeof(instanceInfoT) );
    inst->sigid  = mti_FindSignal( "/top/s1" );
    inst->procid = mti_CreateProcess( "sigWatcher", watchSignal, inst );
  }
```

HDL code

```
    entity for_model is
    end for_model;

    architecture a of for_model is
      attribute foreign of a : architecture is "initForeign for_model.sl";
    begin
    end a;

    library ieee;
    use ieee.std_logic_1164.all;

    entity top is
    end top;

    architecture a of top is

      signal s1 : std_logic := '0';

      component for_model is
      end component;

      for all : for_model use entity work.for_model(a);

    begin

      s1 <= not s1 after 5 ns;

      i1 : for_model;

    end a;
```

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,0] delta 0: Signal /top/s1 is '0'
# Time [0,5] delta 0: Signal /top/s1 is '1'
# Time [0,10] delta 0: Signal /top/s1 is '0'
# Time [0,15] delta 0: Signal /top/s1 is '1'
# Time [0,20] delta 0: Signal /top/s1 is '0'
# Time [0,25] delta 0: Signal /top/s1 is '1'
# Time [0,30] delta 0: Signal /top/s1 is '0'
# Turning off signal watcher.
VSIM 2> quit
```

Related Topics

mti_VsimFree()

mti_TickLength()

mti_GetAttrFromVsimTestrecord()

Returns an attribute from a test record of a UCDB database.

Syntax

int = mti_GetAttrFromVsimTestrecord(key, value)

Arguments

Name	Type	Description
key	const char *	Name of the attribute whose value is accessed.
value	void *	Value of the specified attribute.

Return Values

- 0 — upon successful retrieval of attribute
- -1 — otherwise

Description

Gets the value of the specified attribute from the vsim implicit test data record.

mti_GetArrayElementType()

Gets the type of an array type's subelements.

Syntax

elem_type = mti_GetArrayElementType(array_type)

Arguments

Name	Type	Description
array_type	mtiTypeIdT	A type ID for a VHDL or SystemC array type

Return Values

Name	Type	Description
elem_type	mtiTypeIdT	The type ID for the subelements of the array

Description

mti_GetArrayElementType() returns a handle to the type ID for the subelements of the specified array type. If the array_type parameter is not a handle to an array type, then NULL is returned.

Examples

FLI code

```
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  char * typestr;

  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:  typestr = "Scalar";  break;
    case MTI_TYPE_ARRAY:  typestr = "Array";  break;
    case MTI_TYPE_RECORD:  typestr = "Record";  break;
    case MTI_TYPE_ENUM:  typestr = "Enum";  break;
    case MTI_TYPE_PHYSICAL:  typestr = "Physical";  break;
    case MTI_TYPE_REAL:  typestr = "Real";  break;
    case MTI_TYPE_ACCESS:  typestr = "Access";  break;
    case MTI_TYPE_FILE:  typestr = "File";  break;
    case MTI_TYPE_TIME:  typestr = "Time";  break;
    default:  typestr = "UNKNOWN";  break;
  }

  return typestr;
}

static void printSignalInfo( mtiSignalIdT sigid, int indent )
{
  char         * fullname;
  int            i;
  mtiInt32T      num_elems;
  mtiSignalIdT * elem_list;
  mtiTypeIdT     sig_type;
  mtiTypeIdT     elem_type;

  fullname = mti_GetSignalNameIndirect( sigid, 0, 0 );
  sig_type = mti_GetSignalType( sigid );

  mti_PrintFormatted( "\n%*cSignal %s is of type %s.\n", indent, ' ',
                    fullname, getTypeStr( sig_type ));
  mti_VsimFree( fullname );

  elem_type = mti_GetArrayElementType( sig_type );
  if ( elem_type ) {
    mti_PrintFormatted( "%*cIts subelements are of type %s.\n",
                        indent, ' ', getTypeStr( elem_type ));
  } else {
    if ( mti_GetTypeKind( sig_type ) == MTI_TYPE_RECORD ) {
      mti_PrintFormatted( "%*cThe record subelements are:\n",
                          indent, ' ' );
      elem_list = mti_GetSignalSubelements( sigid, 0 );
      num_elems = mti_GetNumRecordElements( sig_type );
      for ( i = 0; i < num_elems; i++ ) {
        printSignalInfo( elem_list[i], indent+2 );
      }
      mti_VsimFree( elem_list );
    } else {
      mti_PrintFormatted( "%*cThere are no array subelements.\n",
```

```
                              indent, ' ' );
      }
    }
  }

  static void initInstance( void * param )
  {
    mtiSignalIdT sigid;

    for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
          sigid; sigid = mti_NextSignal() ) {
        printSignalInfo( sigid, 1 );
    }
  }

  void initForeign(
    mtiRegionIdT       region,   /* The ID of the region in which this    */
                                 /* foreign architecture is instantiated. */
    char              *param,    /* The last part of the string in the    */
                                 /* foreign attribute.                    */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
  )
  {
    mti_AddLoadDoneCB( initInstance, 0 );
  }
```

HDL code

```
e
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )      of integer;

  type rectype is record
    a : real;
    b : std_logic;
    c : bitarray;
  end record;

end top;

architecture a of top is

  signal bitsig1      : bit        := '1';
  signal stdlogicsig1 : std_logic := '1';

  signal bitarr1      : bitarray  := "0110";
  signal stdlogicarr1 : std_logic_vector( 1 to 4 ) := "01LH";
  signal intarr1      : intarray  := ( 10, 11, 12 );
  signal strarr1      : string(1 to 5) := "hello";

  signal rec1         : rectype   := ( 1.2, '0', "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig1      <= not bitsig1 after 5 ns;
  stdlogicsig1 <= not stdlogicsig1 after 5 ns;

  bitarr1      <= not bitarr1 after 5 ns;

  intarr1(1)   <= intarr1(1) + 1 after 5 ns;
  intarr1(2)   <= intarr1(2) + 1 after 5 ns;
  intarr1(3)   <= intarr1(3) + 1 after 5 ns;

  stdlogicarr1 <= not stdlogicarr1 after 5 ns;
```

```
    strarr1      <= "there" after 10 ns;

  rec1.a <= rec1.a + 1.1 after 5 ns;
  rec1.b <= not rec1.b after 5 ns;
  rec1.c <= not rec1.c after 5 ns;

 nd a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
#  Signal bitsig1 is of type Enum.
#  There are no array subelements.
#
#  Signal stdlogicsig1 is of type Enum.
#  There are no array subelements.
#
#  Signal bitarr1 is of type Array.
#  Its subelements are of type Enum.
#
#  Signal stdlogicarr1 is of type Array.
#  Its subelements are of type Enum.
#
#  Signal intarr1 is of type Array.
#  Its subelements are of type Scalar.
#
#  Signal strarr1 is of type Array.
#  Its subelements are of type Enum.
#
#  Signal rec1 is of type Record.
#  The record subelements are:
#
#    Signal rec1.a is of type Real.
#    There are no array subelements.
#
#    Signal rec1.b is of type Enum.
#    There are no array subelements.
#
#    Signal rec1.c is of type Array.
#    Its subelements are of type Enum.
VSIM 1> run 10
VSIM 2> quit
```

Related Topics

mti_VsimFree()

mti_TickLength()

mti_GetArraySignalValue()

Gets the value of a signal of type array.

Syntax

value = mti_GetArraySignalValue(signal_id, buffer)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal of type array
buffer	void *	A buffer into which the value is to be placed; OPTIONAL - can be NULL

Return Values

Name	Type	Description
value	void *	A pointer to the value of the specified signal

Description

mti_GetArraySignalValue() returns the value of an array-type signal.

If the buffer parameter is NULL, then mti_GetArraySignalValue() allocates memory for the value and returns a pointer to it. The caller is responsible for freeing the returned pointer with mti_VsimFree(). If the buffer parameter is not NULL, then mti_GetArraySignalValue() copies the value into the buffer parameter and also returns a pointer to it. The appropriate length of the buffer parameter can be determined by calling mti_TickLength() on the type of the array signal.

The array value is interpreted as follows:

For a subelement of type	The value should be cast to
Enum	(char *) if <= 256 values
	(mtiInt32T *) if > 256 values
Physical	(mtiInt32T *)
Real	(double *)
Scalar (Integer)	(mtiInt32T *)
Time	(mtiTime64T *)

Examples

FLI code

```
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;     /* List of signals. */
  mtiProcessIdT  proc;         /* Test process id. */
} instanceInfoT;

static void printValue( mtiSignalIdT sigid, mtiTypeIdT sigtype, int indent )
{
  switch ( mti_GetTypeKind(sigtype) ) {
   case MTI_TYPE_ENUM:
   case MTI_TYPE_PHYSICAL:
   case MTI_TYPE_SCALAR:
    {
     mtiInt32T scalar_val;
     scalar_val = mti_GetSignalValue( sigid );
     mti_PrintFormatted( "  %d\n", scalar_val );
    }
    break;
   case MTI_TYPE_ARRAY:
    {
     int          i;
     mtiInt32T     num_elems;
     mtiTypeIdT    elem_type;
     mtiTypeKindT  elem_typekind;
     void        * array_val;

     array_val = mti_GetArraySignalValue( sigid, 0 );
     num_elems = mti_TickLength( sigtype );
     elem_type = mti_GetArrayElementType( sigtype );
     elem_typekind = mti_GetTypeKind( elem_type );
     switch ( elem_typekind ) {
      case MTI_TYPE_ENUM:
        {
         char ** enum_values;
         enum_values = mti_GetEnumValues( elem_type );
         if ( mti_TickLength( elem_type ) > 256 ) {
          mtiInt32T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
           mti_PrintFormatted( "  %s", enum_values[val[i]] );
          }
         } else {
          char * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
           mti_PrintFormatted( "  %s", enum_values[val[i]] );
          }
         }
```

```
     }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
       mtiInt32T * val = array_val;
       for ( i = 0; i < num_elems; i++ ) {
        mti_PrintFormatted( "  %d", val[i] );
       }
      }
      break;
    case MTI_TYPE_ARRAY:
     mti_PrintMessage( "  ARRAY" );
      break;
    case MTI_TYPE_RECORD:
     mti_PrintMessage( "  RECORD" );
      break;
    case MTI_TYPE_REAL:
      {
        double * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( "  %g", val[i] );
        }
      }
      break;
    case MTI_TYPE_TIME:
      {
        mtiTime64T * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( "  [%d,%d]",
                            MTI_TIME64_HI32(val[i]),
                            MTI_TIME64_LO32(val[i]) );
        }
      }
      break;
    default:
      break;
   }
  mti_PrintFormatted( "\n" );
  mti_VsimFree( array_val );
 }
 break;
case MTI_TYPE_RECORD:
 {
  int            i;
  mtiSignalIdT * elem_list;
  mtiInt32T      num_elems;
  elem_list = mti_GetSignalSubelements( sigid, 0 );
  num_elems = mti_GetNumRecordElements( sigtype );
  mti_PrintFormatted( "\n" );
  for ( i = 0; i < num_elems; i++ ) {
   mti_PrintFormatted( "%*c", indent, ' ' );
   printValue( elem_list[i], mti_GetSignalType(elem_list[i]),
              indent+2 );
  }
  mti_VsimFree( elem_list );
 }
 break;
```

```
       case MTI_TYPE_REAL:
        {
         double real_val;
         mti_GetSignalValueIndirect( sigid, &real_val );
         mti_PrintFormatted( "  %g\n", real_val );
        }
        break;
       case MTI_TYPE_TIME:
        {
         mtiTime64T time_val;
         mti_GetSignalValueIndirect( sigid, &time_val );
         mti_PrintFormatted( "  [%d,%d]\n",
                            MTI_TIME64_HI32(time_val),
                            MTI_TIME64_LO32(time_val) );
        }
        break;
       default:
        mti_PrintMessage( "\n" );
        break;
      }
    }

    static void checkValues( void *inst_info )
    {
      instanceInfoT *inst_data = (instanceInfoT *)inst_info;
      signalInfoT   *siginfo;

      mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

      for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
        mti_PrintFormatted( "  Signal %s:", siginfo->name );
        printValue( siginfo->sigid, siginfo->typeid, 4 );
      }

      mti_ScheduleWakeup( inst_data->proc, 5 );
    }

    static signalInfoT * setupSignal( mtiSignalIdT sigid )
    {
      signalInfoT * siginfo;

      siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
      siginfo->sigid  = sigid;
      siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
      siginfo->typeid = mti_GetSignalType( sigid );
      siginfo->next   = 0;

      return( siginfo );
    }

    static void initInstance( void * param )
    {
      instanceInfoT * inst_data;
      mtiSignalIdT    sigid;
      signalInfoT   * curr_info;
      signalInfoT   * siginfo;

      inst_data           = mti_Malloc( sizeof(instanceInfoT) );
```

```
      inst_data->sig_info = 0;

      for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
            sigid; sigid = mti_NextSignal() ) {
        siginfo = setupSignal( sigid );
        if ( inst_data->sig_info == 0 ) {
            inst_data->sig_info = siginfo;
        }
        else {
            curr_info->next = siginfo;
        }
        curr_info = siginfo;
      }

      inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                    (void *)inst_data );
      mti_ScheduleWakeup( inst_data->proc, 6 );
}
void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char               *param,   /* The last part of the string in the    */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  signal bitsig      : bit        := '1';
  signal intsig      : integer    := 21;
  signal realsig     : real       := 16.35;
  signal timesig     : time       := 5 ns;
  signal stdlogicsig : std_logic  := 'H';

  signal bitarr      : bitarray   := "0110";
  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
  signal intarr      : intarray   := ( 10, 11, 12 );
  signal realarr     : realarray  := ( 11.6, 101.22 );
  signal timearr     : timearray  := ( 15 ns, 6 ns );

  signal rec         : rectype    := ( '0', 1, 3.7, 'H', "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig      <= not bitsig after 5 ns;
  intsig      <= intsig + 1 after 5 ns;
  realsig     <= realsig + 1.5 after 5 ns;
```

```
    timesig     <= timesig + 1 ns after 5 ns;
    stdlogicsig <= not stdlogicsig after 5 ns;

    bitarr      <= not bitarr after 5 ns;

    intarr(1)   <= intarr(1) + 1 after 5 ns;
    intarr(2)   <= intarr(2) + 1 after 5 ns;
    intarr(3)   <= intarr(3) + 1 after 5 ns;

    realarr(1)  <= realarr(1) + 0.5 after 5 ns;
    realarr(2)  <= realarr(2) + 0.5 after 5 ns;

    timearr(-1) <= timearr(-1) + 1 ns after 5 ns;
    timearr(0)  <= timearr(0)  + 1 ns after 5 ns;

    stdlogicarr <= not stdlogicarr after 5 ns;

    rec.a       <= not rec.a after 5 ns;
    rec.b       <= rec.b + 1 after 5 ns;
    rec.c       <= rec.c + 2.5 after 5 ns;
    rec.d       <= not rec.d after 5 ns;
    rec.e       <= not rec.e after 5 ns;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 10
# Time [0,6]:
#   Signal bitsig:  0
#   Signal intsig:  22
#   Signal realsig:  17.85
#   Signal timesig:  [0,6]
#   Signal stdlogicsig:  2
#   Signal bitarr: '1'  '0'  '0'  '1'
#   Signal stdlogicarr: '1'  '0'  '1'  '0'
#   Signal intarr:  11  12  13
#   Signal realarr:  12.1  101.72
#   Signal timearr:  [0,16]  [0,7]
#   Signal rec:
#       1
#       2
#       6.2
#       2
#       '0'  '1'  '1'  '0'
VSIM 2> run 10
# Time [0,11]:
#   Signal bitsig:  1
#   Signal intsig:  23
#   Signal realsig:  19.35
#   Signal timesig:  [0,7]
#   Signal stdlogicsig:  3
#   Signal bitarr: '0'  '1'  '1'  '0'
#   Signal stdlogicarr: '0'  '1'  '0'  '1'
#   Signal intarr:  12  13  14
#   Signal realarr:  12.6  102.22
#   Signal timearr:  [0,17]  [0,8]
#   Signal rec:
#       0
#       3
#       8.7
#       3
#       '1'  '0'  '0'  '1'
# Time [0,16]:
#   Signal bitsig:  0
#   Signal intsig:  24
#   Signal realsig:  20.85
#   Signal timesig:  [0,8]
#   Signal stdlogicsig:  2
#   Signal bitarr: '1'  '0'  '0'  '1'
#   Signal stdlogicarr: '1'  '0'  '1'  '0'
#   Signal intarr:  13  14  15
```

```
#    Signal realarr:  13.1   102.72
#    Signal timearr:  [0,18]   [0,9]
#    Signal rec:
#        1
#        4
#        11.2
#        2
#        '0'  '1'  '1'  '0'
VSIM 3> quit
```

Related Topics

mti_TickLength()

mti_VsimFree()

mti_GetArrayVarValue()

Gets the value of a VHDL or SystemC variable of type array.

Syntax

value = mti_GetArrayVarValue(variable_id, buffer)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL or SystemC variable of type array
buffer	void *	A buffer into which the value is to be placed; OPTIONAL - can be NULL

Return Values

Name	Type	Description
value	void *	A pointer to the value of the specified variable

Description

mti_GetArrayVarValue() returns the value of an array-type VHDL variable.

If the buffer parameter is NULL, then mti_GetArrayVarValue() returns a pointer to the value, which should be treated as read-only data. (Changing the value pointed to by this pointer actually changes the variable's value.) This pointer must not be freed.

If the buffer parameter is not NULL, then mti_GetArrayVarValue() copies the value into the buffer parameter and also returns a pointer to it. The appropriate length of the buffer parameter can be determined by calling mti_TickLength() on the type of the array variable.

The array value is interpreted as follows:

For a subelement of type	The value should be cast to
Enum	(char *) if <= 256 values
	(mtiInt32T *) if > 256 values
Physical	(mtiInt32T *)
Real	(double *)
Scalar (Integer)	(mtiInt32T *)
Time	(mtiTime64T *)

Examples

FLI code

```
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag * next;
  char                * name;
  mtiSignalIdT          varid;
  mtiTypeIdT            typeid;
} varInfoT;

typedef struct {
  varInfoT   * var_info;    /* List of variables. */
  mtiProcessIdT   proc;     /* Test process id. */
} instanceInfoT;

static void printValue( mtiVariableIdT varid, mtiTypeIdT vartype, int indent )
{
  switch ( mti_GetTypeKind(vartype) ) {
   case MTI_TYPE_ENUM:
   case MTI_TYPE_PHYSICAL:
   case MTI_TYPE_SCALAR:
     {
      mtiInt32T scalar_val;
      scalar_val = mti_GetVarValue( varid );
      mti_PrintFormatted( "  %d\n", scalar_val );
     }
    break;
   case MTI_TYPE_ARRAY:
     {
      int            i;
      mtiInt32T      num_elems;
      mtiTypeIdT     elem_type;
      mtiTypeKindT   elem_typekind;
      void         * array_val;

      array_val = mti_GetArrayVarValue( varid, 0 );
      num_elems = mti_TickLength( vartype );
      elem_type = mti_GetArrayElementType( vartype );
      elem_typekind = mti_GetTypeKind( elem_type );
      switch ( elem_typekind ) {
       case MTI_TYPE_ENUM:
         {
          char ** enum_values;
          enum_values = mti_GetEnumValues( elem_type );
          if ( mti_TickLength( elem_type ) > 256 ) {
           mtiInt32T * val = array_val;
           for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %s", enum_values[val[i]] );
           }
          } else {
           char * val = array_val;
           for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %s", enum_values[val[i]] );
           }
          }
```

```
       }
        break;
      case MTI_TYPE_PHYSICAL:
      case MTI_TYPE_SCALAR:
       {
        mtiInt32T * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( "  %d", val[i] );
        }
       }
        break;
      case MTI_TYPE_ARRAY:
       mti_PrintMessage( "  ARRAY" );
        break;
      case MTI_TYPE_RECORD:
       mti_PrintMessage( "  RECORD" );
        break;
      case MTI_TYPE_REAL:
       {
         double * val = array_val;
         for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  %g", val[i] );
         }
       }
        break;
      case MTI_TYPE_TIME:
       {
         mtiTime64T * val = array_val;
         for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  [%d,%d]",
                              MTI_TIME64_HI32(val[i]),
                              MTI_TIME64_LO32(val[i]) );
         }
       }
        break;
      default:
        break;
     }
    mti_PrintFormatted( "\n" );
   }
  break;
 case MTI_TYPE_RECORD:
  {
   int            i;
   mtiVariableIdT * elem_list;
   mtiInt32T       num_elems;
   elem_list = mti_GetVarSubelements( varid, 0 );
   num_elems = mti_GetNumRecordElements( vartype );
   mti_PrintFormatted( "\n" );
   for ( i = 0; i < num_elems; i++ ) {
    mti_PrintFormatted( "%*c", indent, ' ' );
    printValue( elem_list[i], mti_GetVarType(elem_list[i]),
               indent+2 );
   }
   mti_VsimFree( elem_list );
  }
  break;
 case MTI_TYPE_REAL:
```

```
        {
         double real_val;
         mti_GetVarValueIndirect( varid, &real_val );
         mti_PrintFormatted( "  %g\n", real_val );
        }
        break;
      case MTI_TYPE_TIME:
        {
         mtiTime64T time_val;
         mti_GetVarValueIndirect( varid, &time_val );
         mti_PrintFormatted( "  [%d,%d]\n",
                             MTI_TIME64_HI32(time_val),
                             MTI_TIME64_LO32(time_val) );
        }
        break;
      default:
        mti_PrintMessage( "\n" );
        break;
     }
}

static void checkValues( void *inst_info )
{
   instanceInfoT *inst_data = (instanceInfoT *)inst_info;
   varInfoT   *varinfo;

   mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

   for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
     mti_PrintFormatted( " Variable %s:", varinfo->name );
     printValue( varinfo->varid, varinfo->typeid, 4 );
   }

   mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable( mtiVariableIdT varid )
{
   varInfoT * varinfo;

   varinfo         = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
   varinfo->varid  = varid;
   varinfo->name   = mti_GetVarName( varid );
   varinfo->typeid = mti_GetVarType( varid );
   varinfo->next   = 0;

   return( varinfo );
}

static void initInstance( void * param )
{
   instanceInfoT * inst_data;
   mtiProcessIdT   procid;
   mtiVariableIdT  varid;
   varInfoT     * curr_info;
   varInfoT     * varinfo;

   inst_data            = mti_Malloc( sizeof(instanceInfoT) );
```

```
        inst_data->var_info = 0;

        for ( procid = mti_FirstProcess( mti_GetTopRegion() );
              procid; procid = mti_NextProcess() ) {
         for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
          varinfo = setupVariable( varid );
          if ( inst_data->var_info == 0 ) {
              inst_data->var_info = varinfo;
          }
          else {
              curr_info->next = varinfo;
          }
          curr_info = varinfo;
         }
        }

        inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                             (void *)inst_data );
       mti_ScheduleWakeup( inst_data->proc, 6 );
    }
    void initForeign(
      mtiRegionIdT        region,   /* The ID of the region in which this    */
                                    /* foreign architecture is instantiated. */
      char                *param,   /* The last part of the string in the    */
                                    /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
    )
    {
      mti_AddLoadDoneCB( initInstance, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process
    variable bitsig      : bit       := '1';
    variable intsig      : integer   := 21;
    variable realsig     : real      := 16.35;
    variable timesig     : time      := 5 ns;
    variable stdlogicsig : std_logic := 'H';

    variable bitarr      : bitarray  := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray  := ( 10, 11, 12 );
    variable realarr     : realarray := ( 11.6, 101.22 );
    variable timearr     : timearray := ( 15 ns, 6 ns );

    variable rec         : rectype   := ( '0', 1, 3.7, 'H', "1001" );

  begin
    bitsig      := not bitsig;
```

```
        intsig     := intsig + 1;
        realsig    := realsig + 1.5;
        timesig    := timesig + 1 ns;
        stdlogicsig := not stdlogicsig;

        bitarr     := not bitarr;

        intarr(1)  := intarr(1) + 1;
        intarr(2)  := intarr(2) + 1;
        intarr(3)  := intarr(3) + 1;

        realarr(1) := realarr(1) + 0.5;
        realarr(2) := realarr(2) + 0.5;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0)  + 1 ns;

        stdlogicarr := not stdlogicarr;

        rec.a      := not rec.a;
        rec.b      := rec.b + 1;
        rec.c      := rec.c + 2.5;
        rec.d      := not rec.d;
        rec.e      := not rec.e;

        wait for 5 ns;

    end process;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,6]:
#    Variable bitsig:  1
#    Variable intsig:  23
#    Variable realsig:  19.35
#    Variable timesig:  [0,7]
#    Variable stdlogicsig:  3
#    Variable bitarr:  '0'  '1'  '1'  '0'
#    Variable stdlogicarr:  '0'  '1'  '0'  '1'
#    Variable intarr:  12  13  14
#    Variable realarr:  12.6  102.22
#    Variable timearr:  [0,17]  [0,8]
#    Variable rec:
#        0
#        3
#        8.7
#        3
#        '1'  '0'  '0'  '1'
# Time [0,11]:
#    Variable bitsig:  0
#    Variable intsig:  24
#    Variable realsig:  20.85
#    Variable timesig:  [0,8]
#    Variable stdlogicsig:  2
#    Variable bitarr:  '1'  '0'  '0'  '1'
#    Variable stdlogicarr:  '1'  '0'  '1'  '0'
#    Variable intarr:  13  14  15
#    Variable realarr:  13.1  102.72
#    Variable timearr:  [0,18]  [0,9]
#    Variable rec:
#        1
#        4
#        11.2
#        2
#        '0'  '1'  '1'  '0'
VSIM 2> quit
```

Related Topics

mti_VsimFree()

mti_GetCallingRegion()

Gets the current elaboration region during elaboration or the region of the currently active process or signal resolution function or the current environment during simulation.

Syntax

region_id = mti_GetCallingRegion()

Arguments

None

Return Values

Name	Type	Description
region_id	mtiRegionIdT	A handle to the calling region

Description

During elaboration, mti_GetCallingRegion() returns the region ID of the current elaboration region. During simulation, mti_GetCallingRegion() returns the region ID of the currently active process or signal resolution function context. If there is currently no active process or signal resolution function, mti_GetCallingRegion() returns the region ID of the current environment set by the environment command.

A foreign subprogram can call mti_GetCallingRegion() to determine the region in which the process containing the subprogram call resides.

The region ID returned by mti_GetCallingRegion() can be a VHDL, Verilog, or SystemC region. You can use a handle to a Verilog region with PLI functions to obtain information about or access objects in the Verilog region.

At the beginning of time zero and during the time that the Load Done callback functions are called, the calling region is the last non-foreign region that was elaborated.

Examples

FLI code

```
#include <mti.h>

void doProc( void )
{
  char *          region_name;
  mtiRegionIdT    regid;

  regid = mti_GetCallingRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Time [%d,%d]: doProc Procedure: "
                      "Calling region is %s\n",
                      mti_NowUpper(), mti_Now(), region_name );
  mti_VsimFree( region_name );
}

static void checkEnv( void )
{
  char *          region_name;
  mtiRegionIdT    regid;

  regid = mti_GetCallingRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Time [%d,%d]: checkEnv Function: "
                      "Calling region is %s\n",
                      mti_NowUpper(), mti_Now(), region_name );
  mti_VsimFree( region_name );
}

static void checkRegion( void )
{
  char *          region_name;
  mtiRegionIdT    regid;

  regid = mti_GetCallingRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Time [%d,%d]: checkRegion Function: "
                      "Calling region is %s\n",
                      mti_NowUpper(), mti_Now(), region_name );
  mti_VsimFree( region_name );
}

static void initInstance( void * param )
{
  char *          region_name;
  mtiProcessIdT   procid;
  mtiRegionIdT    regid;

  regid = mti_GetCallingRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Load Done Callback Function: Calling region is %s\n",
                      region_name );
  mti_VsimFree( region_name );

  procid = mti_CreateProcess( "Test Process", checkRegion, 0 );
```

```
        mti_ScheduleWakeup( procid, 10 );
    }

    void initForeign(
      mtiRegionIdT       region,   /* The ID of the region in which this     */
                                   /* foreign architecture is instantiated.  */
      char              *param,    /* The last part of the string in the     */
                                   /* foreign attribute.                     */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
    )
    {
      char *        region_name;
      mtiRegionIdT regid;

      mti_PrintFormatted( "Foreign Init Function:\n" );

      region_name = mti_GetRegionFullName( region );
      mti_PrintFormatted( "  Region parameter is %s\n", region_name );
      mti_VsimFree( region_name );

      regid = mti_GetCallingRegion();
      region_name = mti_GetRegionFullName( regid );
      mti_PrintFormatted( "  Calling region is %s\n", region_name );
      mti_VsimFree( region_name );

      mti_AddLoadDoneCB( initInstance, 0 );
      mti_AddEnvCB( checkEnv, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

package for_pkg is

   procedure test_proc;
   attribute foreign of test_proc : procedure is "doProc for_model.sl;";

end for_pkg;

package body for_pkg is

   procedure test_proc is
     begin
     end;

end for_pkg;

use work.for_pkg.all;

entity lower is
end lower;

architecture level of lower is
begin

  p1 : process
    begin
       test_proc;
       wait for 15 ns;
    end process;

end level;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

   component for_model
   end component;

   for all : for_model use entity work.for_model(a);

   component lower
   end component;

   for all : lower use entity work.lower(level);
```

```
    begin

      linst1 : lower;

      linst2 : lower;

      finst  : for_model;

    end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.for_pkg(body)
    # Loading ./for_model.sl
    # Loading work.top(a)
    # Loading work.lower(level)
    # Loading work.for_model(a)
    # Foreign Init Function:
    #   Region parameter is /top/finst
    #   Calling region is /top/finst
    # Time [0,0]: checkEnv Function: Calling region is /top/linst2
    # Load Done Callback Function: Calling region is /top/linst2
    VSIM 1> run 0
    # Time [0,0]: doProc Procedure: Calling region is /top/linst2
    # Time [0,0]: doProc Procedure: Calling region is /top/linst1
    VSIM 2> env
    # sim:/top
    VSIM 3> env finst
    # Time [0,0]: checkEnv Function: Calling region is /top/finst
    # sim:/top/finst
    VSIM 4> run 10
    # Time [0,10]: checkRegion Function: Calling region is /top
    VSIM 5> env
    # sim:/top/finst
    VSIM 6> env /top
    # Time [0,10]: checkEnv Function: Calling region is /top
    # sim:/top
    VSIM 7> run 10
    # Time [0,15]: doProc Procedure: Calling region is /top/linst2
    # Time [0,15]: doProc Procedure: Calling region is /top/linst1
    VSIM 8> env
    # sim:/top
    VSIM 9> env linst1
    # Time [0,20]: checkEnv Function: Calling region is /top/linst1
    # sim:/top/linst1
    VSIM 10> run 15
    # Time [0,30]: doProc Procedure: Calling region is /top/linst2
    # Time [0,30]: doProc Procedure: Calling region is /top/linst1
    VSIM 11> quit
```

Related Topics

mti_VsimFree()

mti_GetCheckpointFilename()

Gets the name of the current checkpoint file.

Syntax

filename = mti_GetCheckpointFilename()

Arguments

None

Return Values

Name	Type	Description
filename	char *	A pointer to the name of the current checkpoint file

Description

mti_GetCheckpointFilename() returns the filename specified with the most recent checkpoint or restore command. A NULL is returned if no checkpoint or restore command has been given.

You must not free the returned pointer.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
  mti_PrintFormatted( "Checkpoint filename is %s\n",
                     mti_GetCheckpointFilename() );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Checkpoint filename is %s\n",
                     mti_GetCheckpointFilename() );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);
begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint my_checkpoint.file
# Saving instance info "for_model"
# Checkpoint filename is my_checkpoint.file
VSIM 3> run 30
VSIM 4> checkpoint my_other_cp.file
# Saving instance info "for_model"
# Checkpoint filename is my_other_cp.file
VSIM 5> run 40
VSIM 6> restore my_checkpoint.file
# Loading checkpoint/restore data from file "my_checkpoint.file"
# Checkpoint created Fri Jun 23 10:18:12 2000
# Restoring state at time 20 ns, iteration 1
# Restored instance info "for_model"
# Checkpoint filename is my_checkpoint.file
VSIM 7> run 10
VSIM 8> quit
# Cleaning up...
```

mti_GetCheckpointDirname()

Returns the directory into which the checkpoint file is being written.

Syntax

dirname = mti_GetCheckpointDirname()

Arguments

None

Return Values

Name	Type	Description
dirname	char *	Directory name into which the checkpoint file is being written.

Description

This call is only valid if you specified the -d argument to the checkpoint command.

mti_GetCurrentRegion()

Gets the current elaboration region during elaboration or the current environment during simulation.

Syntax

region_id = mti_GetCurrentRegion()

Arguments

None

Return Values

Name	Type	Description
region_id	mtiRegionIdT	A handle to the current region

Description

During elaboration, mti_GetCurrentRegion() returns the region ID of the current elaboration region. During simulation, mti_GetCurrentRegion() returns the region ID of the current environment set by the environment command.

The region ID returned by mti_GetCurrentRegion() can be either a VHDL, Verilog, or SystemC region. A handle to a Verilog region can be used with PLI functions to obtain information about or access objects in the Verilog region.

Examples

FLI code

```
#include <mti.h>

void doProc( void )
{
  char *          region_name;
  mtiRegionIdT    regid;

  regid = mti_GetCurrentRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Time [%d,%d]: doProc Procedure: "
                      "Current region is %s\n",
                      mti_NowUpper(), mti_Now(), region_name );
  mti_VsimFree( region_name );
}

static void checkEnv( void )
{
  char *          region_name;
  mtiRegionIdT    regid;

  regid = mti_GetCurrentRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Time [%d,%d]: checkEnv Function: "
                      "Current region is %s\n",
                      mti_NowUpper(), mti_Now(), region_name );
  mti_VsimFree( region_name );
}

static void checkRegion( void )
{
  char *          region_name;
  mtiRegionIdT    regid;

  regid = mti_GetCurrentRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Time [%d,%d]: checkRegion Function: "
                      "Current region is %s\n",
                      mti_NowUpper(), mti_Now(), region_name );
  mti_VsimFree( region_name );
}

static void initInstance( void * param )
{
  char *          region_name;
  mtiProcessIdT   procid;
  mtiRegionIdT    regid;

  regid = mti_GetCurrentRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "Load Done Callback Function: Current region is %s\n",
                      region_name );
  mti_VsimFree( region_name );

  procid = mti_CreateProcess( "Test Process", checkRegion, 0 );
```

```
    mti_ScheduleWakeup( procid, 10 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char             *param,     /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  char *      region_name;
  mtiRegionIdT regid;

  mti_PrintFormatted( "Foreign Init Function:\n" );

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "  Region parameter is %s\n", region_name );
  mti_VsimFree( region_name );

  regid = mti_GetCurrentRegion();
  region_name = mti_GetRegionFullName( regid );
  mti_PrintFormatted( "  Current region is %s\n", region_name );
  mti_VsimFree( region_name );

  mti_AddLoadDoneCB( initInstance, 0 );
  mti_AddEnvCB( checkEnv, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

package for_pkg is

  procedure test_proc;
  attribute foreign of test_proc : procedure is "doProc for_model.sl;";

end for_pkg;

package body for_pkg is

  procedure test_proc is
    begin
    end;

end for_pkg;

use work.for_pkg.all;

entity lower is
end lower;

architecture level of lower is
begin

  p1 : process
    begin
      test_proc;
      wait for 15 ns;
    end process;

end level;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

  component lower
  end component;

  for all : lower use entity work.lower(level);
```

```
begin

   linst1 : lower;

   linst2 : lower;

   finst  : for_model;

end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.for_pkg(body)
    # Loading ./for_model.sl
    # Loading work.top(a)
    # Loading work.lower(level)
    # Loading work.for_model(a)
    # Foreign Init Function:
    #    Region parameter is /top/finst
    #    Current region is /top/finst
    # Time [0,0]: checkEnv Function: Current region is /top
    # Load Done Callback Function: Current region is /top
    VSIM 1> run 0
    # Time [0,0]: doProc Procedure: Current region is /top
    # Time [0,0]: doProc Procedure: Current region is /top
    VSIM 2> env
    # sim:/top
    VSIM 3> env finst
    # Time [0,0]: checkEnv Function: Current region is /top/finst
    # sim:/top/finst
    VSIM 4> run 10
    # Time [0,10]: checkRegion Function: Current region is /top/finst
    VSIM 5> env
    # sim:/top/finst
    VSIM 6> env /top
    # Time [0,10]: checkEnv Function: Current region is /top
    # sim:/top
    VSIM 7> run 10
    # Time [0,15]: doProc Procedure: Current region is /top
    # Time [0,15]: doProc Procedure: Current region is /top
    VSIM 8> env
    # sim:/top
    VSIM 9> env linst1
    # Time [0,20]: checkEnv Function: Current region is /top/linst1
    # sim:/top/linst1
    VSIM 10> run 15
    # Time [0,30]: doProc Procedure: Current region is /top/linst1
    # Time [0,30]: doProc Procedure: Current region is /top/linst1
    VSIM 11> env
    # sim:/top/linst1
    VSIM 12> quit
```

Related Topics

mti_VsimFree()

mti_GetDriverNames()

Gets the names of all drivers on a VHDL signal.

Syntax

driver_names = mti_GetDriverNames(signal_id, length)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL signal
length	mtiInt32T *	Returns the number of names in the returned name array

Return Values

Name	Type	Description
driver_names	char **	A pointer to a NULL-terminated array of the names of the signal's drivers

Description

mti_GetDriverNames() returns a NULL-terminated array of the names of the drivers that are driving values onto the specified signal. The simulator returns the number of names in the array in the length parameter. If there is an error, or if the signal is in a nodebug region, or if the type of the signal is not a scalar enumeration type, then the simulator sets the length parameter to zero and returns a NULL. You must not free the returned array and character strings. The driver names are valid only until the next call to mti_GetDriverNames().

You can use mti_GetDriverNames() in conjunction with mti_GetDriverValues() because the arrays returned from each function are in the same order; therefore, each driver name can be associated with a value.

mti_GetDriverNames() returns the same information as the driver name part of the output of the drivers command.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  char * signame;
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;
} instanceInfoT;

char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}

void checkSignal( void * param )
{
  char          ** drv_names;
  char           * drv_values;
  instanceInfoT * inst = (instanceInfoT*)param;
  int              i;
  mtiInt32T        names_length;
  mtiInt32T        sigval;
  mtiInt32T        values_length;

  sigval = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d:\n Signal %s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      inst->signame, convertStdLogicValue( sigval ) );
```

```
   mti_PrintFormatted( " Drivers:\n" );
   drv_names = mti_GetDriverNames( inst->sigid, &names_length );
   drv_values = mti_GetDriverValues( inst->sigid, &values_length );
   for ( i = 0; i < names_length; i++ ) {
       mti_PrintFormatted( "  %s : %s\n",
                           convertStdLogicValue(drv_values[i]),
                           drv_names[i] );
   }

   mti_ScheduleWakeup( inst->procid, 5 );
}


void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}


void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instanceInfoT * inst;

  inst           = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
  inst->sigid    = mti_FindSignal( "/top/s1" );
  inst->signame  = mti_GetSignalName( inst->sigid );
  inst->procid   = mti_CreateProcess( "checkSignal", checkSignal, inst );
  mti_ScheduleWakeup( inst->procid, 1 );
  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity lower is
  port ( pt : INOUT std_logic := '0' );
end lower;

architecture a of lower is
begin
  p0 : process
    begin
      pt <= '1';
      wait for 5 ns;
      pt <= 'L';
      wait for 5 ns;
      pt <= 'W';
      wait for 5 ns;
    end process;
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

  component lower
    port ( pt : INOUT std_logic );
  end component;

begin

  p1 : process
    begin
      s1 <= 'H';
      wait for 5 ns;
      s1 <= 'L';
      wait for 5 ns;
      s1 <= 'X';
      wait for 5 ns;
    end process;

  p2 : process
    begin
      s1 <= '1';
      wait for 5 ns;
      s1 <= '0';
      wait for 5 ns;
      s1 <= 'W';
      wait for 5 ns;
    end process;
```

```
      inst1 : lower port map ( s1 );

   end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.lower(a)
# Loading ./for_model.sl
VSIM 1> run 1
# Time [0,1] delta 0:
#   Signal s1 is '1'
#   Drivers:
#    '1' : /top/inst1/p0
#    '1' : /top/p2
#    'H' : /top/p1
VSIM 2> drivers /top/s1
# Drivers for /top/s1:
#  1 : Signal /top/s1
#     1 : Driver /top/inst1/p0
#     1 : Driver /top/p2
#     H : Driver /top/p1
#
VSIM 3> run 5
# Time [0,6] delta 0:
#   Signal s1 is '0'
#   Drivers:
#    'L' : /top/inst1/p0
#    '0' : /top/p2
#    'L' : /top/p1
VSIM 4> drivers /top/s1
# Drivers for /top/s1:
#  0 : Signal /top/s1
#     L : Driver /top/inst1/p0
#     0 : Driver /top/p2
#     L : Driver /top/p1
#
VSIM 5> run 5
# Time [0,11] delta 0:
#   Signal s1 is 'X'
#   Drivers:
#    'W' : /top/inst1/p0
#    'W' : /top/p2
#    'X' : /top/p1
VSIM 6> drivers /top/s1
# Drivers for /top/s1:
#  X : Signal /top/s1
#     W : Driver /top/inst1/p0
#     W : Driver /top/p2
#     X : Driver /top/p1
#
VSIM 7> quit
# Cleaning up...
```

Related Topics

mti_VsimFree()

mti_GetDriverSubelements()

Gets the subelements of a composite driver.

Syntax

driver_list = mti_GetDriverSubelements(driver_id, buffer)

Arguments

Name	Type	Description
driver_id	mtiDriverIdT	A handle to an array-type driver
buffer	mtiDriverIdT *	A buffer into which the subelement driver IDs are to be placed; OPTIONAL - can be NULL

Return Values

Name	Type	Description
driver_list	mtiDriverIdT *	A pointer to an array of driver IDs for each of the subelements of the specified array-type driver or NULL if the specified driver is not of an array type

Description

mti_GetDriverSubelements() returns an array of driver IDs for each of the subelements of the specified array-type driver.

If the buffer parameter is NULL, then mti_GetDriverSubelements() allocates memory for the value and returns a pointer to it. The caller is responsible for freeing the returned pointer with mti_VsimFree().

If the buffer parameter is not NULL, then mti_GetDriverSubelements() copies the value into the buffer parameter and also returns the buffer parameter.

Examples

FLI code

```c
#include <stdlib.h>

#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  mtiSignalIdT   sigid;
  mtiDriverIdT * drv_elems;
  int            index;
  int            num_elems;
} instanceInfoT;

char * convertStdLogicValue( char sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}
void driveSignal( void * param )
{
  char          * region_name;
  char          * sigval;
  instanceInfoT * inst = (instanceInfoT*)param;
  int             i;

  region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid));
  sigval      = (char *)mti_GetArraySignalValue( inst->sigid, 0 );
  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is {",
                  mti_NowUpper(), mti_Now(), mti_Delta(),
                  region_name, mti_GetSignalName( inst->sigid ) );
  for ( i = 0; i < inst->num_elems; i++ ) {
```

```
      mti_PrintFormatted( " %s", convertStdLogicValue( sigval[i] ) );
    }
    mti_PrintFormatted( " }\n" );

    switch ( sigval[inst->index] ) {
      case STD_LOGIC_U:  sigval[inst->index] = STD_LOGIC_X;  break;
      case STD_LOGIC_X:  sigval[inst->index] = STD_LOGIC_0;  break;
      case STD_LOGIC_0:  sigval[inst->index] = STD_LOGIC_1;  break;
      case STD_LOGIC_1:  sigval[inst->index] = STD_LOGIC_Z;  break;
      case STD_LOGIC_Z:  sigval[inst->index] = STD_LOGIC_W;  break;
      case STD_LOGIC_W:  sigval[inst->index] = STD_LOGIC_L;  break;
      case STD_LOGIC_L:  sigval[inst->index] = STD_LOGIC_H;  break;
      case STD_LOGIC_H:  sigval[inst->index] = STD_LOGIC_D;  break;
      case STD_LOGIC_D:  sigval[inst->index] = STD_LOGIC_U;  break;
      default:  sigval[inst->index] = STD_LOGIC_U;  break;
    }
    mti_ScheduleDriver( inst->drv_elems[inst->index], sigval[inst->index],
                        5, MTI_INERTIAL );

    inst->index++;
      if ( inst->index >= inst->num_elems ) {
      inst->index = 0;
    }
    mti_VsimFree( region_name );
    mti_VsimFree( sigval );
  }

  void cleanupCallback( void * param )
  {
    mti_PrintMessage( "Cleaning up...\n" );
    free( param );
  }

  void initForeign(
    mtiRegionIdT        region,   /* The ID of the region in which this     */
                                  /* foreign architecture is instantiated.  */
    char               *param,    /* The last part of the string in the     */
                                  /* foreign attribute.                     */
    mtiInterfaceListT *generics,  /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports      /* A list of ports for the foreign model.  */
  )
  {
    instanceInfoT * inst;
    mtiDriverIdT    drvid;
    mtiProcessIdT   procid;

    inst          = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
    inst->sigid   = mti_FindSignal( "/top/s1" );
    drvid         = mti_CreateDriver( inst->sigid );
    inst->drv_elems = mti_GetDriverSubelements( drvid, 0 );
    inst->num_elems = mti_TickLength( mti_GetSignalType( inst->sigid ));
    inst->index   = 0;
    procid        = mti_CreateProcess( "sigDriver", driveSignal, inst );
    mti_Sensitize( procid, inst->sigid, MTI_EVENT );
    mti_ScheduleWakeup( procid, 0 );
    mti_SetDriverOwner( drvid, procid );
    mti_AddQuitCB( cleanupCallback, inst );
    mti_AddRestartCB( cleanupCallback, inst );
```

```
      }
```

HDL code

```
    library ieee;
    use ieee.std_logic_1164.all;

    entity top is
    end top;

    architecture a of top is
      signal s1 : std_logic_vector( 3 downto 0 ) := "0000";
    begin
    end a;
```

Simulation output

```
    % vsim -c top -foreign "initForeign for_model.sl"
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -foreign {initForeign for_model.sl} -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.top(a)
    # Loading ./for_model.sl
    VSIM 1> run 50
    # Time [0,0] delta 1: Signal /top/s1 is { '0' '0' '0' '0' }
    # Time [0,5] delta 0: Signal /top/s1 is { '1' '0' '0' '0' }
    # Time [0,10] delta 0: Signal /top/s1 is { '1' '1' '0' '0' }
    # Time [0,15] delta 0: Signal /top/s1 is { '1' '1' '1' '0' }
    # Time [0,20] delta 0: Signal /top/s1 is { '1' '1' '1' '1' }
    # Time [0,25] delta 0: Signal /top/s1 is { 'Z' '1' '1' '1' }
    # Time [0,30] delta 0: Signal /top/s1 is { 'Z' 'Z' '1' '1' }
    # Time [0,35] delta 0: Signal /top/s1 is { 'Z' 'Z' 'Z' '1' }
    # Time [0,40] delta 0: Signal /top/s1 is { 'Z' 'Z' 'Z' 'Z' }
    # Time [0,45] delta 0: Signal /top/s1 is { 'W' 'Z' 'Z' 'Z' }
    # Time [0,50] delta 0: Signal /top/s1 is { 'W' 'W' 'Z' 'Z' }
    VSIM 2> quit
    # Cleaning up...
```

mti_GetDriverValues()

Gets the values of all drivers on a VHDL signal.

Syntax

value = mti_GetDriverValues(signal_id, length)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL signal
length	mtiInt32T *	Returns the number of elements in the returned value array

Return Values

Name	Type	Description
value	char *	A pointer to a statically allocated array of std_logic driver values

Description

mti_GetDriverValues() returns the values of drivers for the specified signal. The returned pointer is a pointer to statically allocated memory; therefore, you must not free this pointer. The simulator returns the array element count in the length parameter. If there is an error, or if the signal is in a nodebug region, or if the type of the signal is not a scalar enumeration type, then the simulator sets the length parameter to zero and returns no values. The values in the array are overwritten on each call to mti_GetDriverValues().

You can use mti_GetDriverValues() can be used in conjunction with mti_GetDriverNames() because the arrays returned from each function are in the same order; therefore, each driver value can be associated with a name.

mti_GetDriverValues() returns the same information as the driver value part of the output of the drivers command.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  char * signame;
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;
} instanceInfoT;

char * convertStdLogicValue( char sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default: retval = "?";  break;
  }
  return retval;
}

void checkSignal( void * param )
{
  char          ** drv_names;
  char           * drv_values;
  instanceInfoT * inst = (instanceInfoT*)param;
  int              i;
  mtiInt32T        names_length;
  mtiInt32T        sigval;
  mtiInt32T        values_length;

  sigval = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d:\n Signal %s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      inst->signame, convertStdLogicValue( sigval ) );
```

```
    mti_PrintFormatted( " Drivers:\n" );
    drv_names = mti_GetDriverNames( inst->sigid, &names_length );
    drv_values = mti_GetDriverValues( inst->sigid, &values_length );
    for ( i = 0; i < names_length; i++ ) {
        mti_PrintFormatted( "  %s : %s\n",
                            convertStdLogicValue(drv_values[i]),
                            drv_names[i] );
    }

    mti_ScheduleWakeup( inst->procid, 5 );
}


void cleanupCallback( void * param )
{
    mti_PrintMessage( "Cleaning up...\n" );
    free( param );
}


void initForeign(
    mtiRegionIdT      region,   /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
    char             *param,    /* The last part of the string in the    */
                                /* foreign attribute.                    */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
    instanceInfoT * inst;

    inst           = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
    inst->sigid    = mti_FindSignal( "/top/s1" );
    inst->signame  = mti_GetSignalName( inst->sigid );
    inst->procid   = mti_CreateProcess( "checkSignal", checkSignal, inst );
    mti_ScheduleWakeup( inst->procid, 1 );
    mti_AddQuitCB( cleanupCallback, inst );
    mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity lower is
  port ( pt : INOUT std_logic := '0' );
end lower;

architecture a of lower is
begin
  p0 : process
    begin
      pt <= '1';
      wait for 5 ns;
      pt <= 'L';
      wait for 5 ns;
      pt <= 'W';
      wait for 5 ns;
    end process;
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

  component lower
    port ( pt : INOUT std_logic );
  end component;

begin

  p1 : process
    begin
      s1 <= 'H';
      wait for 5 ns;
      s1 <= 'L';
      wait for 5 ns;
      s1 <= 'X';
      wait for 5 ns;
    end process;

  p2 : process
    begin
      s1 <= '1';
      wait for 5 ns;
      s1 <= '0';
      wait for 5 ns;
      s1 <= 'W';
      wait for 5 ns;
    end process;
```

```
    inst1 : lower port map ( s1 );

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.lower(a)
# Loading ./for_model.sl
VSIM 1> run 1
# Time [0,1] delta 0:
#  Signal s1 is '1'
#  Drivers:
#   '1' : /top/inst1/p0
#   '1' : /top/p2
#   'H' : /top/p1
VSIM 2> drivers /top/s1
# Drivers for /top/s1:
#  1 : Signal /top/s1
#    1 : Driver /top/inst1/p0
#    1 : Driver /top/p2
#    H : Driver /top/p1
#
VSIM 3> run 5
# Time [0,6] delta 0:
#  Signal s1 is '0'
#  Drivers:
#   'L' : /top/inst1/p0
#   '0' : /top/p2
#   'L' : /top/p1
VSIM 4> drivers /top/s1
# Drivers for /top/s1:
#  0 : Signal /top/s1
#    L : Driver /top/inst1/p0
#    0 : Driver /top/p2
#    L : Driver /top/p1
#
VSIM 5> run 5
# Time [0,11] delta 0:
#  Signal s1 is 'X'
#  Drivers:
#   'W' : /top/inst1/p0
#   'W' : /top/p2
#   'X' : /top/p1
VSIM 6> drivers /top/s1
# Drivers for /top/s1:
#  X : Signal /top/s1
#    W : Driver /top/inst1/p0
#    W : Driver /top/p2
#    X : Driver /top/p1
#
VSIM 7> quit
# Cleaning up...
```

mti_GetDrivingSignals()

Gets a handle to all of the VHDL or SystemC signals driving a signal.

Syntax

signal_list = mti_GetDrivingSignals(signal_name)

Arguments

Name	Type	Description
signal_name	char *	The name of the signal for which the driving signals are to be found

Return Values

Name	Type	Description
signal_list	mtiSignalIdT *	A NULL-terminated array of driving signal IDs for the specified signal or NULL if there is an error or no drivers are found

Description

mti_GetDrivingSignals() returns a NULL-terminated array of driving signal IDs for the specified signal. The signal is specified by name using either a full hierarchical name or a relative name. A relative name is relative to the region set by the environment command. The default is the top-level VHDL or SystemC region.

The caller is responsible for freeing the returned pointer with mti_VsimFree().

mti_GetDrivingSignals() returns the same signal IDs as those used by the drivers command to generate the signal part of its output.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  char *         signame;
  mtiSignalIdT  sigid;
  mtiProcessIdT procid;
} instanceInfoT;

char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}

void checkSignal( void * param )
{
  char           * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  int              i;
  mtiInt32T       sigval;
  mtiSignalIdT  * drv_signals;

  sigval = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d:\n Signal %s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      inst->signame, convertStdLogicValue( sigval ) );

  mti_PrintFormatted( " Driving Signals for %s:\n", inst->signame );
```

```
    drv_signals = mti_GetDrivingSignals( inst->signame );
    for ( i = 0; drv_signals[i]; i++ ) {
     region_name =
       mti_GetRegionFullName( mti_GetSignalRegion( drv_signals[i] ));
     mti_PrintFormatted( "      %s/%s\n", region_name,
                        mti_GetSignalName( drv_signals[i] ) );
     mti_VsimFree( region_name );
    }
    mti_ScheduleWakeup( inst->procid, 5 );
  }

  void cleanupCallback( void * param )
  {
    mti_PrintMessage( "Cleaning up...\n" );
    free( param );
  }
  void initForeign(
    mtiRegionIdT      region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
    char             *param,   /* The last part of the string in the    */
                               /* foreign attribute.                    */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
  )
  {
    instanceInfoT * inst;

    inst            = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
    inst->signame   = "/top/s1";
    inst->sigid     = mti_FindSignal( inst->signame );
    inst->procid    = mti_CreateProcess( "checkSignal", checkSignal, inst );
    mti_ScheduleWakeup( inst->procid, 1 );
    mti_AddQuitCB( cleanupCallback, inst );
    mti_AddRestartCB( cleanupCallback, inst );
  }
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity lower is
  port ( pt1 : OUT std_logic := '0';
         pt2 : IN  std_logic
       );
end lower;

architecture a of lower is
begin

  pt1 <= pt2 after 5 ns;

end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';
  signal s2 : std_logic := '0';

  component lower
    port ( pt1 : OUT std_logic;
           pt2 : IN  std_logic
         );
  end component;

begin

  s2 <= not s2 after 5 ns;

  s1 <= s2 after 5 ns;

  p1 : process
    begin
      s1 <= 'H';
      wait for 5 ns;
      s1 <= 'L';
      wait for 5 ns;
      s1 <= 'W';
      wait for 5 ns;
    end process;

  inst1 : lower port map ( s1, s2 );

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.lower(a)
# Loading ./for_model.sl
VSIM 1> run 1
# Time [0,1] delta 0:
#   Signal /top/s1 is '0'
#   Driving Signals for /top/s1:
#       /top/s1
VSIM 2> drivers /top/s1
# Drivers for /top/s1:
#  0 : Signal /top/s1
#     0 : Driver /top/inst1/line__14
#     H : Driver /top/p1
#     0 : Driver /top/line__39
#
VSIM 3> run 5
# Time [0,6] delta 0:
#   Signal /top/s1 is '0'
#   Driving Signals for /top/s1:
#       /top/s1
VSIM 4> drivers /top/s1
# Drivers for /top/s1:
#  0 : Signal /top/s1
#     0 : Driver /top/inst1/line__14
#         1 at 10 ns
#     L : Driver /top/p1
#     0 : Driver /top/line__39
#         1 at 10 ns
#
VSIM 5> quit
# Cleaning up...
```

mti_GetEnumValues()

Gets the values of an enumeration type.

Syntax

enum_values = mti_GetEnumValues(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC enumeration type

Return Values

Name	Type	Description
enum_values	char **	A pointer to an array of enumeration literals for the specified enumeration type or NULL if the specified type is not an enumeration type

Description

mti_GetEnumValues() returns a pointer to an array of enumeration literals for the specified enumeration type or NULL if the specified type is not an enumeration type. You must not free the returned pointer. You can find the number of elements in the array by calling mti_TickLength() on the enumeration type. The first element in the array is the left-most value of the enumeration type.

Examples

Array Looping Example

Suppose you have VHDL enumerated types: days, week, and reverse. Your FLI code might contain a signal or variable whose mtiTypeIdTvariable, my_type, is of type week or reverse. The for loop uses mti_TickLow() and mti_TickHigh() to access the array elements of enum — the values of the enumerated type named my_type.

VHDL Enumerated Types (code snippet):

```
Type days is (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
Subtype week is days range Mon to Fri;
Subtype reverse is days Fri downto Mon;
```

FLI code snippet

```
mtiTypeIdT my_type = mti_GetSignalType(sigid); /* type of the signal */
/* or */
mtiTypeIdT my_type = mti_GetVarType(varid); /* type of the variable */
char ** enums = mti_GetEnumValues(my_type);
int i;

for (i = mti_TickLow(my_type); i <= mti_TickHigh(my_type); i++) {
    mti_PrintFormatted("enum %d is %s\n", i, enums[i]);
}
```

FLI code

```
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;      /* List of signals. */
  mtiProcessIdT  proc;          /* Test process id. */
} instanceInfoT;

static void printValue( mtiSignalIdT sigid, mtiTypeIdT sigtype, int indent )
{
  switch ( mti_GetTypeKind(sigtype) ) {
  case MTI_TYPE_ENUM:
    {
     char **   enum_values;
     mtiInt32T scalar_val;
     scalar_val  = mti_GetSignalValue( sigid );
     enum_values = mti_GetEnumValues( sigtype );
     mti_PrintFormatted( "  %s\n", enum_values[scalar_val] );
    }
   break;
  case MTI_TYPE_PHYSICAL:
  case MTI_TYPE_SCALAR:
    {
     mtiInt32T scalar_val;
     scalar_val = mti_GetSignalValue( sigid );
     mti_PrintFormatted( "  %d\n", scalar_val );
    }
   break;
  case MTI_TYPE_ARRAY:
    {
     int          i;
     mtiInt32T    num_elems;
     mtiTypeIdT   elem_type;
     mtiTypeKindT elem_typekind;
     void       * array_val;

     array_val = mti_GetArraySignalValue( sigid, 0 );
     num_elems = mti_TickLength( sigtype );
     elem_type = mti_GetArrayElementType( sigtype );
     elem_typekind = mti_GetTypeKind( elem_type );
     switch ( elem_typekind ) {
      case MTI_TYPE_ENUM:
        {
          char ** enum_values;
          enum_values = mti_GetEnumValues( elem_type );
          if ( mti_TickLength( elem_type ) > 256 ) {
           mtiInt32T * val = array_val;
           for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %s", enum_values[val[i]] );
           }
```

```
          } else {
           char * val = array_val;
           for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %s", enum_values[val[i]] );
           }
          }
         }
         break;
        case MTI_TYPE_PHYSICAL:
        case MTI_TYPE_SCALAR:
         {
          mtiInt32T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
           mti_PrintFormatted( "  %d", val[i] );
          }
         }
         break;
        case MTI_TYPE_ARRAY:
         mti_PrintMessage( "  ARRAY" );
         break;
        case MTI_TYPE_RECORD:
         mti_PrintMessage( "  RECORD" );
         break;
        case MTI_TYPE_REAL:
         {
          double * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
           mti_PrintFormatted( "  %g", val[i] );
          }
         }
         break;
        case MTI_TYPE_TIME:
         {
          mtiTime64T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
           mti_PrintFormatted( "  [%d,%d]",
                               MTI_TIME64_HI32(val[i]),
                               MTI_TIME64_LO32(val[i]) );
          }
         }
         break;
        default:
         break;
       }
      mti_PrintFormatted( "\n" );
      mti_VsimFree( array_val );
     }
     break;
    case MTI_TYPE_RECORD:
     {
      int              i;
      mtiSignalIdT * elem_list;
      mtiInt32T       num_elems;
      elem_list = mti_GetSignalSubelements( sigid, 0 );
      num_elems = mti_GetNumRecordElements( sigtype );
      mti_PrintFormatted( "\n" );
      for ( i = 0; i < num_elems; i++ ) {
       mti_PrintFormatted( "%*c", indent, ' ' );
```

```
        printValue( elem_list[i], mti_GetSignalType(elem_list[i]),
                    indent+2 );
      }
      mti_VsimFree( elem_list );
     }
     break;
   case MTI_TYPE_REAL:
    {
     double real_val;
     mti_GetSignalValueIndirect( sigid, &real_val );
     mti_PrintFormatted( "  %g\n", real_val );
    }
    break;
   case MTI_TYPE_TIME:
    {
     mtiTime64T time_val;
     mti_GetSignalValueIndirect( sigid, &time_val );
     mti_PrintFormatted( "  [%d,%d]\n",
                         MTI_TIME64_HI32(time_val),
                         MTI_TIME64_LO32(time_val) );
    }
    break;
   default:
    mti_PrintMessage( "\n" );
    break;
  }
}

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  signalInfoT   *siginfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
    mti_PrintFormatted( "  Signal %s:", siginfo->name );
    printValue( siginfo->sigid, siginfo->typeid, 4 );
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo        = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid = sigid;
  siginfo->name  = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid = mti_GetSignalType( sigid );
  siginfo->next  = 0;

  return( siginfo );
}

static void initInstance( void * param )
{
```

```
            instanceInfoT * inst_data;
            mtiSignalIdT    sigid;
            signalInfoT  * curr_info;
            signalInfoT  * siginfo;

            inst_data          = mti_Malloc( sizeof(instanceInfoT) );
            inst_data->sig_info = 0;

            for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
                  sigid; sigid = mti_NextSignal() ) {
              siginfo = setupSignal( sigid );
              if ( inst_data->sig_info == 0 ) {
                inst_data->sig_info = siginfo;
              }
              else {
                curr_info->next = siginfo;
              }
              curr_info = siginfo;
            }

            inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                                 (void *)inst_data );
            mti_ScheduleWakeup( inst_data->proc, 6 );
          }
          void initForeign(
            mtiRegionIdT       region,   /* The ID of the region in which this     */
                                         /* foreign architecture is instantiated.  */
            char             *param,     /* The last part of the string in the     */
                                         /* foreign attribute.                     */
            mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
            mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
          )
          {
            mti_AddLoadDoneCB( initInstance, 0 );
          }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;

  type rectype is record
    a : bit;
    b : std_logic;
    c : bitarray;
  end record;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal stdlogicsig : std_logic := 'H';
  signal bitarr      : bitarray  := "0110";
  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

  signal rec         : rectype   := ( '0', 'H', "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig      <= not bitsig after 5 ns;
  stdlogicsig <= not stdlogicsig after 5 ns;

  bitarr      <= not bitarr after 5 ns;
  stdlogicarr <= not stdlogicarr after 5 ns;

  rec.a       <= not rec.a after 5 ns;
  rec.b       <= not rec.b after 5 ns;
  rec.c       <= not rec.c after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,6]:
#   Signal bitsig:  '0'
#   Signal stdlogicsig:  '0'
#   Signal bitarr:  '1'  '0'  '0'  '1'
#   Signal stdlogicarr:  '1'  '0'  '1'  '0'
#   Signal rec:
#       '1'
#       '0'
#       '0'  '1'  '1'  '0'
# Time [0,11]:
#   Signal bitsig:  '1'
#   Signal stdlogicsig:  '1'
#   Signal bitarr:  '0'  '1'  '1'  '0'
#   Signal stdlogicarr:  '0'  '1'  '0'  '1'
#   Signal rec:
#       '0'
#       '1'
#       '1'  '0'  '0'  '1'
VSIM 2> quit
```

mti_GetEquivSignal()

Finds the representation of the signal according to the simulator.

Syntax

result = mti_GetEquivSignal(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the VHDL or SystemC signal or NULL if the signal is not found

Return Values

Name	Type	Description
result	mtiSignalIdT	If a SystemC handle or a handle to a composite VHDL signal the return value is the signal.
		If a scalar VHDL signal, the return value is the signal handle for what the simulator considers to be a representative signal.

Description

This function determines if two scalar signals are internally represented by the same simulation signal. If two signals have the same internal simulation signal they are electrically equivalent. Two electrically equivalent signals are not guaranteed to have the same internal simulation signal.

To determine if two signals are the "same" in the simulator, use the comparison:

```
mti_GetEquivSignal(signal1) == mti_GetEquivSignal(signal2)
```

mti_GetGenericList()

Gets a list of the VHDL generics defined for a region.

Syntax

generic_list = mti_GetGenericList(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL region

Return Values

Name	Type	Description
generic_list	mtiInterfaceListT *	A pointer to a NULL-terminated list of generics for the specified region or NULL if there are no generics in the specified region

Description

mti_GetGenericList() returns a NULL-terminated list of the generics defined for the specified region. This list is in the same interface format as the C initialization function generics list. The caller is responsible for freeing each element in the list with mti_Free().

If there are no generics in the region, then mti_GetGenericList() returns NULL.

Examples

FLI code

```
#include "mti.h"

void printGenericList( mtiInterfaceListT * generic_list, int free_it )
{
  mtiInterfaceListT * glp;
  mtiInterfaceListT * glp_next;

  for ( glp = generic_list; glp; glp = glp_next ) {
   mti_PrintFormatted( "    %s =", glp->name );
   switch ( mti_GetTypeKind( glp->type ) ) {
    case MTI_TYPE_ENUM:
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
     mti_PrintFormatted( " %d\n", glp->u.generic_value );
     break;
    case MTI_TYPE_REAL:
     mti_PrintFormatted( " %g\n", glp->u.generic_value_real );
     break;
    case MTI_TYPE_TIME:
     mti_PrintFormatted( " [%d,%d]\n",
                         MTI_TIME64_HI32(glp->u.generic_value_time),
                          MTI_TIME64_LO32(glp->u.generic_value_time) );
     break;
    case MTI_TYPE_ARRAY:
     {
      int        i;
      mtiInt32T  num_elems = mti_TickLength( glp->type );
      mtiTypeIdT elem_type = mti_GetArrayElementType( glp->type );

      switch ( mti_GetTypeKind( elem_type ) ) {
       case MTI_TYPE_PHYSICAL:
       case MTI_TYPE_SCALAR:
        {
         mtiInt32T * val = glp->u.generic_array_value;
         for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( " %d", val[i] );
         }
        }
        break;
       case MTI_TYPE_ARRAY:
        mti_PrintFormatted( " ARRAY of ARRAYs" );
        break;
       case MTI_TYPE_RECORD:
        mti_PrintFormatted( " ARRAY of RECORDs" );
        break;
       case MTI_TYPE_ENUM:
        {
         char ** enum_values = mti_GetEnumValues( elem_type );
         char *  array_val = glp->u.generic_array_value;

         for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( " %s",
                              enum_values[array_val[i]] );
         }
```

```
       }
       break;
      case MTI_TYPE_REAL:
       {
        double * val = glp->u.generic_array_value;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( " %g", val[i] );
        }
       }
       break;
      case MTI_TYPE_TIME:
       {
        mtiTime64T * val = glp->u.generic_array_value;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( " [%d,%d]",
                             MTI_TIME64_HI32(val[i]),
                             MTI_TIME64_LO32(val[i]) );
        }
       }
       break;
      default:
       break;
     }
     mti_PrintFormatted( "\n" );
     }
     break;
    default:
     mti_PrintFormatted( "\n" );
     break;
    }
    glp_next = glp->nxt;
    if ( free_it ) {
     mti_Free( glp );
    }
   }
}

void printRegionInfo( char * region_name )
{
  mtiInterfaceListT * generic_list;
  mtiRegionIdT        regid;

  regid = mti_FindRegion( region_name );
  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    mti_PrintFormatted( "  Region %s:\n", region_name );
    mti_VsimFree( region_name );
    generic_list = mti_GetGenericList( regid );
    printGenericList( generic_list, 1 );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printRegionInfo( "top" );
  printRegionInfo( "inst1" );
  printRegionInfo( "inst1/i1" );
```

```
      printRegionInfo( "inst1/flip" );
      printRegionInfo( "/top/inst1/toggle" );
   }

   void initForeign(
     mtiRegionIdT       region,   /* The ID of the region in which this    */
                                  /* foreign architecture is instantiated.  */
     char              *param,   /* The last part of the string in the     */
                                  /* foreign attribute.                     */
     mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
     mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
   )
   {
     mti_AddLoadDoneCB( loadDoneCB, 0 );

     mti_PrintMessage( "\nElaboration phase:\n" );
     mti_PrintMessage( "  Foreign function generics:\n" );
     printGenericList( generics, 0 );
   }
```

HDL code

```
package my_pkg is
  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

  type intarray     is array( 1 to 3 ) of integer;
  type realarray    is array( 0 to 2 ) of real;
  type timearray    is array( 2 to 4 ) of time;
  type bigtimearray is array( 1 to 3 ) of bigtime;
end my_pkg;

entity for_model is
  generic ( whoami : string := "Do not know" );
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "+initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( min_delay : time := 5 ns;
            max_delay : time := 10 ns );
  port ( a : in bit;
         b : out bit );
end inv;

architecture b of inv is
begin
  b <= a after min_delay;
end b;

use work.my_pkg.all;

entity mid is
  generic ( g1 : bit := '0';
            g2 : integer := 11;
            g3 : real := 12.97;
            g4 : bit_vector := "0010";
            g5 : intarray := ( 1, 2, 3 );
            g6 : realarray := ( 10.5, 16.8, 21.39 );
            g7 : timearray := ( 3 ns, 18 ns, 123 ns );
            g8 : bigtime := 13 hour;
            g9 : bigtimearray := ( 2 hour, 4 hour, 6 hour ) );
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
```

```
   signal s4 : bit := '0';

component for_model is
  generic ( whoami : string := "Did not say" );
end component;

for all : for_model use entity work.for_model(a);

component inv is
  generic ( min_delay : time := 5 ns;
            max_delay : time := 10 ns );
  port ( a : in bit;
         b : out bit );
  end component;

begin

  flip : inv
    generic map ( 3 ns, 8 ns )
    port map ( s3, s4 );

  s1 <= not s1 after 5 ns;

  toggle : inv port map ( s1, s2 );

  i1 : for_model generic map ( "inst i1" );

end a;

use work.my_pkg.all;

entity top is
end top;

architecture a of top is

  component mid is
    generic ( g1 : bit := '0';
              g2 : integer := 11;
              g3 : real := 12.97;
              g4 : bit_vector := "101";
              g5 : intarray := ( 7, 9, 11 );
              g6 : realarray := ( 8.1, 6.2, 1.34 );
              g7 : timearray := ( 212 ns, 100 ns, 9 ns );
              g8 : bigtime := 40 hour;
              g9 : bigtimearray := ( 8 hour, 16 hour, 32 hour ) );
  end component;

begin

  inst1 : mid generic map ( '1', 42, 101.2, "101101" );

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.my_pkg
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#   Foreign function generics:
#     whoami = 'i' 'n' 's' 't' ' ' 'i' '1'
#
# Load Done phase:
#   Region /top:
#   Region /top/inst1:
#     g1 = 1
#     g2 = 42
#     g3 = 101.2
#     g4 = '1' '0' '1' '1' '0' '1'
#     g5 = 7 9 11
#     g6 = 8.1 6.2 1.34
#     g7 = [0,212] [0,100] [0,9]
#     g8 = 40
#     g9 = 8 16 32
#   Region /top/inst1/i1:
#     whoami = 'i' 'n' 's' 't' ' ' 'i' '1'
#   Region /top/inst1/flip:
#     min_delay = [0,3]
#     max_delay = [0,8]
#   Region /top/inst1/toggle:
#     min_delay = [0,5]
#     max_delay = [0,10]
VSIM 1> run 10
VSIM 2> quit
```

mti_GetLibraryName()

Gets the physical name of the library that contains a region.

Syntax

lib_name = mti_GetLibraryName(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a region

Return Values

Name	Type	Description
lib_name	char *	The physical name of the library that contains the specified design unit region

Description

mti_GetLibraryName() returns the physical name of the library that contains the design unit identified by the specified region. If the region is not a design unit, then the simulator uses the parent design unit. You must not free the returned pointer.

You can use mti_GetLibraryName() on VHDL, Verilog, or SystemC regions.

Examples

FLI code

```c
#include "mti.h"

static void printRegionInfo( mtiRegionIdT regid )
{
  char * lib_name;
  char * region_name;

  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    lib_name    = mti_GetLibraryName( regid );
    mti_PrintFormatted( "  Region %s is in Library %s\n",
                        region_name, lib_name );
    mti_VsimFree( region_name );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printRegionInfo( mti_FindRegion( "top" ) );
  printRegionInfo( mti_FindRegion( "inst1" ) );
  printRegionInfo( mti_FindRegion( "inst1/i1" ) );
  printRegionInfo( mti_FindRegion( "inst1/flip" ) );
  printRegionInfo( mti_FindRegion( "inst1/toggle" ) );
}

void initForeign(
  mtiRegionIdT       region,    /* The ID of the region in which this     */
                                /* foreign architecture is instantiated.  */
  char              *param,     /* The last part of the string in the     */
                                /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  char * lib_name;

  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  lib_name = mti_GetLibraryName( region );
  mti_PrintFormatted( "  Foreign architecture region is in Library %s\n",
                      lib_name );
}
```

HDL code

```
for_model.vhd
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;
inv.vhd
entity inv is
   generic ( delay : time := 5 ns );
   port ( a : in bit;
          b : out bit
        );
end inv;

architecture b of inv is
begin
   b <= a after delay;
end b;
mid.vhd
library for_model_lib;
library inv_lib;

entity mid is
end mid;

architecture a of mid is

   signal s1 : bit := '0';
   signal s2 : bit := '0';
   signal s3 : bit := '0';
   signal s4 : bit := '0';

   component for_model is
   end component;
   for all : for_model use entity for_model_lib.for_model(a);

   component inv is
     generic ( delay : time := 5 ns );
     port ( a : in bit;
            b : out bit
          );
   end component;
   for all : inv use entity inv_lib.inv(b);

begin

   flip : inv port map ( s3, s4 );


   i1 : for_model;

   s1 <= not s1 after 5 ns;

   toggle : inv port map ( s1, s2 );
```

```
end a;
top.vhd
library mid_lib;

entity top is
end top;

architecture a of top is
  component mid is
  end component;
  for all : mid use entity mid_lib.mid(a);
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vlib for_model_lib
% vlib my_inv_lib
% vlib my_mid_lib
% vlib work
% vmap -c
Copying .../modeltech/sunos5/../modelsim.ini to modelsim.ini
% vmap inv_lib my_inv_lib
Modifying modelsim.ini
% vmap mid_lib my_mid_lib
Modifying modelsim.ini
% vcom -93 for_model.vhd -work for_model_lib
Model Technology ModelSim SE vcom 5.5 Compiler 2000.10 Mar  2 2001
-- Loading package standard
-- Compiling entity for_model
-- Compiling architecture a of for_model
% vcom -93 inv.vhd -work inv_lib
Model Technology ModelSim SE vcom 5.5 Compiler 2000.10 Mar  2 2001
-- Loading package standard
-- Compiling entity inv
-- Compiling architecture b of inv
% vcom -93 mid.vhd -work mid_lib
Model Technology ModelSim SE vcom 5.5 Compiler 2000.10 Mar  2 2001
-- Loading package standard
-- Compiling entity mid
-- Compiling architecture a of mid
-- Loading entity for_model
-- Loading entity inv
% vcom -93 top.vhd
Model Technology ModelSim SE vcom 5.5 Compiler 2000.10 Mar  2 2001
-- Loading package standard
-- Compiling entity top
-- Compiling architecture a of top
-- Loading entity mid
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.5

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading my_mid_lib.mid(a)
# Loading my_inv_lib.inv(b)
# Loading for_model_lib.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#    Foreign architecture region is in Library for_model_lib
#
# Load Done phase:
#    Region /top is in Library work
#    Region /top/inst1 is in Library my_mid_lib
#    Region /top/inst1/i1 is in Library for_model_lib
#    Region /top/inst1/flip is in Library my_inv_lib
#    Region /top/inst1/toggle is in Library my_inv_lib
VSIM 1> run 10
```

```
VSIM 2> quit
```

mti_GetNextEventTime()

Gets the next event time (from a foreign subprogram or callback).

Syntax

status = mti_GetNextEventTime(next_time)

Arguments

Name	Type	Description
next_time	mtiTime64T *	Returns the time at which the next simulation event will occur (see below for details)

Return Values

Name	Type	Description
status	int	A number that indicates which type of events are pending (see below for details)

Description

mti_GetNextEventTime() returns the next simulation event time when called from within a foreign subprogram or callback function. It always returns the current simulation time when called from within a VHDL process.

The return value and next_time parameter are set as follows:

Status	Description	next_time
0	There are no pending events	current time
1	There are pending events	maturity time of the next pending event
2	There are pending postponed processes for the last delta of the current time	maturity time of the next pending event (which is the current time if there are no future pending events)

Examples

FLI code

```
#include <mti.h>
static void checkTime( void )
{
  int        status;
  mtiTime64T next_time;

  status = mti_GetNextEventTime( &next_time );
  switch ( status ) {
   case 0:
    mti_PrintFormatted( "  No pending events; Next time is [%d,%d]\n",
                        MTI_TIME64_HI32( next_time ),
                        MTI_TIME64_LO32( next_time ) );
    break;
   case 1:
    mti_PrintFormatted( "  Pending events; Next time is [%d,%d]\n",
                        MTI_TIME64_HI32( next_time ),
                        MTI_TIME64_LO32( next_time ) );
    break;
   case 2:
    mti_PrintFormatted( "  Pending postponed processes; "
                        "Next time is [%d,%d]\n",
                        MTI_TIME64_HI32( next_time ),
                        MTI_TIME64_LO32( next_time ) );
    break;
  }
}

void doProc( void )
{
  mti_PrintFormatted( "Time [%d,%d]: doProc()\n",
                      mti_NowUpper(), mti_Now() );
  checkTime();
}

static void checkEnv( void )
{
  mti_PrintFormatted( "Time [%d,%d]: checkEnv()\n",
                      mti_NowUpper(), mti_Now() );
  checkTime();
}

static void checkRegion( void )
{
  mti_PrintFormatted( "Time [%d,%d]: checkRegion()\n",
                      mti_NowUpper(), mti_Now() );
  /*
   * NOTE:  mti_GetNextEventTime() will always return the current
   *        time when called from inside of a VHDL process.
   */
  checkTime();
}

static void initInstance( void * param )
{
```

```
    mtiProcessIdT procid;
    mtiSignalIdT  sigid;

    sigid  = mti_FindSignal( "/top/s1" );
    procid = mti_CreateProcess( "Test Process", checkRegion, 0 );
    mti_Sensitize( procid, sigid, MTI_EVENT );
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this     */
                                /* foreign architecture is instantiated.  */
  char               *param,    /* The last part of the string in the     */
                                /* foreign attribute.                     */
  mtiInterfaceListT *generics,  /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports      /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
  mti_AddEnvCB( checkEnv, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

package for_pkg is
  procedure test_proc;
  attribute foreign of test_proc : procedure is "doProc for_model.sl;";
end for_pkg;

package body for_pkg is
  procedure test_proc is
   begin
   end;
end for_pkg;

use work.for_pkg.all;

entity top is
end top;

architecture a of top is

  component for_model
  end component;
  for all : for_model use entity work.for_model(a);

  signal s1 : bit := '0';

begin

  s1 <= not s1 after 7 ns;

  finst  : for_model;

  p1 : postponed process
   begin
    wait for 15 ns;
    test_proc;
   end process;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.for_pkg(body)
# Loading ./for_model.sl
# Loading work.top(a)
# Loading work.for_model(a)
# Time [0,0]: checkEnv()
#    Pending events; Next time is [0,0]
VSIM 1> run 3
VSIM 2> env finst
# Time [0,3]: checkEnv()
#    Pending events; Next time is [0,7]
# sim:/top/finst
VSIM 3> run 5
# Time [0,7]: checkRegion()
#    Pending events; Next time is [0,7]
VSIM 4> env top
# Time [0,8]: checkEnv()
#    Pending events; Next time is [0,14]
# sim:/top
VSIM 5> run 7
# Time [0,14]: checkRegion()
#    Pending events; Next time is [0,14]
VSIM 6> env finst
# Time [0,15]: checkEnv()
#    Pending postponed processes; Next time is [0,21]
# sim:/top/finst
VSIM 7> quit
```

mti_GetNextNextEventTime()

Gets the next event time (from a VHDL process).

Syntax

status = mti_GetNextNextEventTime(next_time)

Arguments

Name	Type	Description
next_time	mtiTime64T *	Returns the time at which the next simulation event will occur (See below for details)

Return Values

Name	Type	Description
status	int	A number that indicates which types of events are pending (See below for details)

Description

mti_GetNextNextEventTime() returns the next simulation event time when called from within a VHDL process. The stop time of the current run command is considered to be a pending event, as is the stop time of a step command.

The return value and next_time parameter are set as follows:

Status	Description	next_time
0	There are no pending events	current time
1	There are pending events	maturity time of the next pending event
2	There are pending postponed processes for the last delta of the current time	maturity time of the next pending event (which is the current time if there are no future pending events)

Examples

FLI code

```c
#include <mti.h>
static void checkTime( void )
{
  int        status;
  mtiTime64T next_time;

  status = mti_GetNextNextEventTime( &next_time );
  switch ( status ) {
   case 0:
    mti_PrintFormatted( "  No pending events; Next time is [%d,%d]\n",
                        MTI_TIME64_HI32( next_time ),
                        MTI_TIME64_LO32( next_time ) );
    break;
   case 1:
    mti_PrintFormatted( "  Pending events; Next time is [%d,%d]\n",
                        MTI_TIME64_HI32( next_time ),
                        MTI_TIME64_LO32( next_time ) );
    break;
   case 2:
    mti_PrintFormatted( "  Pending postponed processes; "
                        "Next time is [%d,%d]\n",
                        MTI_TIME64_HI32( next_time ),
                        MTI_TIME64_LO32( next_time ) );
    break;
  }
}

void doProc( void )
{
  mti_PrintFormatted( "Time [%d,%d]: doProc()\n",
                      mti_NowUpper(), mti_Now() );
  checkTime();
}

static void checkRegion( void )
{
  mti_PrintFormatted( "Time [%d,%d]: checkRegion()\n",
                      mti_NowUpper(), mti_Now() );
  checkTime();
}

static void initInstance( void * param )
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;

  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "Test Process", checkRegion, 0 );
  mti_Sensitize( procid, sigid, MTI_EVENT );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this   */
                             /* foreign architecture is instantiated. */
```

```
    char              *param,    /* The last part of the string in the     */
                                 /* foreign attribute.                     */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
    mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

package for_pkg is

  procedure test_proc;
  attribute foreign of test_proc : procedure is "doProc for_model.sl;";

end for_pkg;

package body for_pkg is

  procedure test_proc is
   begin
    end;

end for_pkg;

use work.for_pkg.all;

entity top is
end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

  signal s1 : bit := '0';

begin

  s1 <= not s1 after 4 ns;

  finst  : for_model;

  p1 : postponed process
   begin
    wait for 16 ns;
    test_proc;
   end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.for_pkg(body)
# Loading ./for_model.sl
# Loading work.top(a)
# Loading work.for_model(a)
VSIM 1> run 3
VSIM 2> run 4
# Time [0,4]: checkRegion()
#    Pending events; Next time is [0,7]
VSIM 3> run 9
# Time [0,8]: checkRegion()
#    Pending events; Next time is [0,12]
# Time [0,12]: checkRegion()
#    Pending events; Next time is [0,16]
# Time [0,16]: checkRegion()
#    Pending postponed processes; Next time is [0,16]
VSIM 4> quit
```

mti_GetNumRecordElements()

Gets the number of subelements in a VHDL record type.

Syntax

num_elems = mti_GetNumRecordElements(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL record type

Return Values

Name	Type	Description
num_elems	mtiInt32T	The number of subelements in the specified record type

Description

mti_GetNumRecordElements() returns the number of subelements in the specified VHDL record type.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;      /* List of signals. */
  mtiProcessIdT  proc;          /* Test process id. */
} instanceInfoT;

static void printValue( mtiSignalIdT sigid, mtiTypeIdT sigtype, int indent )
{
  switch ( mti_GetTypeKind(sigtype) ) {
   case MTI_TYPE_ENUM:
   case MTI_TYPE_PHYSICAL:
   case MTI_TYPE_SCALAR:
    {
     mtiInt32T scalar_val;
     scalar_val = mti_GetSignalValue( sigid );
     mti_PrintFormatted( "  %d\n", scalar_val );
    }
    break;
   case MTI_TYPE_ARRAY:
    {
     int          i;
     mtiInt32T    num_elems;
     mtiTypeIdT   elem_type;
     mtiTypeKindT elem_typekind;
     void       * array_val;

     array_val = mti_GetArraySignalValue( sigid, 0 );
     num_elems = mti_TickLength( sigtype );
     elem_type = mti_GetArrayElementType( sigtype );
     elem_typekind = mti_GetTypeKind( elem_type );
     switch ( elem_typekind ) {
      case MTI_TYPE_ENUM:
       {
        char ** enum_values;
        enum_values = mti_GetEnumValues( elem_type );
        if ( mti_TickLength( elem_type ) > 256 ) {
         mtiInt32T * val = array_val;
         for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  %s", enum_values[val[i]] );
         }
        } else {
         char * val = array_val;
         for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  %s", enum_values[val[i]] );
         }
        }
```

```
        }
       break;
     case MTI_TYPE_PHYSICAL:
     case MTI_TYPE_SCALAR:
       {
        mtiInt32T * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( "  %d", val[i] );
        }
       }
       break;
     case MTI_TYPE_ARRAY:
      mti_PrintMessage( "  ARRAY" );
       break;
     case MTI_TYPE_RECORD:
      mti_PrintMessage( "  RECORD" );
       break;
     case MTI_TYPE_REAL:
       {
        double * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( "  %g", val[i] );
        }
       }
       break;
     case MTI_TYPE_TIME:
       {
        mtiTime64T * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
         mti_PrintFormatted( "  [%d,%d]",
                             MTI_TIME64_HI32(val[i]),
                             MTI_TIME64_LO32(val[i]) );
        }
       }
       break;
     default:
       break;
    }
   mti_PrintFormatted( "\n" );
   mti_VsimFree( array_val );
  }
 break;
 case MTI_TYPE_RECORD:
  {
   int             i;
   mtiSignalIdT * elem_list;
   mtiInt32T       num_elems;
   elem_list = mti_GetSignalSubelements( sigid, 0 );
   num_elems = mti_GetNumRecordElements( sigtype );
   mti_PrintFormatted( "\n" );
   for ( i = 0; i < num_elems; i++ ) {
    mti_PrintFormatted( "%*cField #%d:", indent, ' ', i+1 );
    printValue( elem_list[i], mti_GetSignalType(elem_list[i]),
                indent+2 );
   }
   mti_VsimFree( elem_list );
  }
  break;
```

```
      case MTI_TYPE_REAL:
       {
        double real_val;
        mti_GetSignalValueIndirect( sigid, &real_val );
        mti_PrintFormatted( "  %g\n", real_val );
       }
       break;
      case MTI_TYPE_TIME:
       {
        mtiTime64T time_val;
        mti_GetSignalValueIndirect( sigid, &time_val );
        mti_PrintFormatted( "  [%d,%d]\n",
                          MTI_TIME64_HI32(time_val),
                          MTI_TIME64_LO32(time_val) );
       }
       break;
      default:
       mti_PrintMessage( "\n" );
       break;
     }
   }

   static void checkValues( void *inst_info )
   {
     instanceInfoT *inst_data = (instanceInfoT *)inst_info;
     signalInfoT   *siginfo;

     mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

     for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
      mti_PrintFormatted( "  Signal %s:", siginfo->name );
      printValue( siginfo->sigid, siginfo->typeid, 4 );
     }

     mti_ScheduleWakeup( inst_data->proc, 5 );
   }

   static signalInfoT * setupSignal( mtiSignalIdT sigid )
   {
     signalInfoT * siginfo;

     siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
     siginfo->sigid  = sigid;
     siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
     siginfo->typeid = mti_GetSignalType( sigid );
     siginfo->next   = 0;

     return( siginfo );
   }

   static void initInstance( void * param )
   {
     instanceInfoT * inst_data;
     mtiSignalIdT    sigid;
     signalInfoT   * curr_info;
     signalInfoT   * siginfo;

     inst_data            = mti_Malloc( sizeof(instanceInfoT) );
```

```
    inst_data->sig_info = 0;

    for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
      sigid; sigid = mti_NextSignal() ) {
     siginfo = setupSignal( sigid );
     if ( inst_data->sig_info == 0 ) {
      inst_data->sig_info = siginfo;
     }
     else {
      curr_info->next = siginfo;
     }
     curr_info = siginfo;
    }

    inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                         (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 6 );
}
void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
  begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic_vector( 7 downto 0 );
    e : bitarray;
  end record;

end top;

architecture a of top is

  signal rec : rectype   := ( '0', 1, 3.7, "10010011", "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  rec.a <= not rec.a after 5 ns;
  rec.b <= rec.b + 1 after 5 ns;
  rec.c <= rec.c + 2.5 after 5 ns;
  rec.d <= not rec.d after 5 ns;
  rec.e <= not rec.e after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,6]:
#   Signal rec:
#     Field #1:  1
#     Field #2:  2
#     Field #3:  6.2
#     Field #4:  '0'  '1'  '1'  '0'  '1'  '1'  '0'  '0'
#     Field #5:  '0'  '1'  '1'  '0'
# Time [0,11]:
#   Signal rec:
#     Field #1:  0
#     Field #2:  3
#     Field #3:  8.7
#     Field #4:  '1'  '0'  '0'  '1'  '0'  '0'  '1'  '1'
#     Field #5:  '1'  '0'  '0'  '1'
VSIM 2> quit
```

mti_GetParentSignal()

Gets the higher up signal to which a signal is connected.

Syntax

parent = mti_GetParentSignal(signal)

Arguments

Name	Type	Description
signal	mtiSignalIdT	A handle to a VHDL or SystemC signal

Return Values

Name	Type	Description
parent	mtiSignalIdT	A handle to the VHDL or SystemC signal higher up in the hierarchy to which the specified signal is connected or NULL if no VHDL or SystemC signal is found

Description

mti_GetParentSignal() returns a handle to the VHDL or SystemC signal higher up in the hierarchy to which the specified IN, OUT, or INOUT signal is connected. It returns a NULL if no higher up VHDL or SystemC signal is found, if the signal is connected through a port mapping which includes a type conversion or conversion function, or if the higher up signal is a Verilog object.

Examples

FLI code

```c
#include <mti.h>

void printSignalInfo( mtiSignalIdT sigid )
{
    char *       signame;
    char *       regname;
    mtiRegionIdT regid;
    mtiSignalIdT parent;

    regid = mti_GetSignalRegion( sigid );
    regname = mti_GetRegionFullName( regid );
    signame = mti_GetSignalNameIndirect( sigid, 0, 0 );
    mti_PrintFormatted( "The parent of %s/%s is ", regname, signame );
    mti_VsimFree( signame );
    mti_VsimFree( regname );
    parent = mti_GetParentSignal( sigid );
    regid = mti_GetSignalRegion( parent );
    regname = mti_GetRegionFullName( regid );
    signame = mti_GetSignalNameIndirect( parent, 0, 0 );
    mti_PrintFormatted( "%s/%s whose parent is ", regname, signame );
    mti_VsimFree( signame );
    mti_VsimFree( regname );
    parent = mti_GetParentSignal( parent );
    if ( parent ) {
        regid = mti_GetSignalRegion( parent );
        regname = mti_GetRegionFullName( regid );
        signame = mti_GetSignalNameIndirect( parent, 0, 0 );
        mti_PrintFormatted( "%s/%s.\n", regname, signame );
        mti_VsimFree( signame );
        mti_VsimFree( regname );
    } else {
        mti_PrintFormatted( "<NULL>.\n" );
    }
}

void loadDoneCB( void * param )
{
    mtiSignalIdT siga;
    mtiSignalIdT sigb;
    mtiSignalIdT sigc;

    siga = mti_FindSignal( "/top/m1/i1/a" );
    sigb = mti_FindSignal( "/top/m1/i1/b" );
    sigc = mti_FindSignal( "/top/m1/i1/c" );

    printSignalInfo( siga );
    printSignalInfo( sigb );
    printSignalInfo( sigc );
}

void initForeign(
    mtiRegionIdT     region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
    char            *param,    /* The last part of the string in the     */
```

```
                                    /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
   )
   {
      mti_AddLoadDoneCB( loadDoneCB, 0 );
   }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit;
         c : in std_logic
       );
end inv;

architecture b of inv is
  signal count : integer := 0;
begin
  b <= a after delay;
  p1 : process( c )
    begin
      count <= count + 1 after 0 ns;
    end process;
end b;

library ieee;
use ieee.std_logic_1164.all;

entity mid is
  generic ( delay : time := 5 ns );
  port ( midin  : in bit;
         midout : out bit_vector(3 downto 0);
         midslv : in std_logic_vector( 7 downto 4 )
       );
end mid;

architecture a of mid is

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit;
           c : in std_logic
         );
  end component;

begin

  i1 : inv port map ( midin, midout(2), midslv(7) );

end a;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : bit_vector( 3 downto 0 ) := "0000";
  signal s2 : bit_vector( 3 downto 0 ) := "0000";
  signal s3 : bit_vector( 3 downto 0 ) := "0000";

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component mid is
    generic ( delay : time := 5 ns );
    port ( midin  : in bit;
           midout : out bit_vector(3 downto 0);
           midslv : in std_logic_vector( 7 downto 4 )
         );
  end component;

begin

  s1(3) <= not s1(3) after 5 ns;

  m1 : mid port map ( s1(3), s2, to_stdlogicvector(s3) );

  f1 : for_model;

end a;
```

Simulation output

The first example shows vsim running in its normal optimization mode. In this case, the simple ports are collapsed for performance and memory efficiency. The immediate parents of the lowest-level signals *a* and *b* are shown to be the top-level signals, and the parent signal to /top/ m1/midslv(7) cannot be found because of the type conversion in the top-level port map.

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
# The parent of /top/m1/i1/a is /top/s1(3) whose parent is <NULL>.
# The parent of /top/m1/i1/b is /top/s2(2) whose parent is <NULL>.
# The parent of /top/m1/i1/c is /top/m1/midslv(7) whose parent is <NULL>.
VSIM 1> quit
```

The second example uses the -nocollapse argument to vsim to cause all ports to be retained so that multiple levels of signal connection are shown. The parent signal to /top/m1/midslv(7) cannot be found because of the type conversion in the top-level port map.

```
% vsim -c -nocollapse top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c -nocollapse top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
# The parent of /top/m1/i1/a is /top/m1/midin whose parent is /top/s1(3).
# The parentof /top/m1/i1/b is /top/m1/midout(2) whose parent is /top/
s2(2).
# The parent of /top/m1/i1/c is /top/m1/midslv(7) whose parent is <NULL>.
VSIM 1> quit
```

mti_GetPhysicalData()

Gets the unit information of a physical type.

Syntax

```
phys_data = mti_GetPhysicalData( type_id )
```

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL physical type

Return Values

Name	Type	Description
phys_data	mtiPhysicalDataT *	A pointer to a linked list of structures each describing the name and position of a unit in the specified physical type

Description

mti_GetPhysicalData() returns a pointer to a linked list of structures each describing the name and position of a unit in the specified physical type. The simulator traverses the linked list by using the next pointer in each structure and a NULL pointer terminates traversal. The caller is responsible for freeing each structure in the list with mti_Free().

mti_GetPhysicalData() returns NULL if the specified type is not a physical type.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                    * name;
  mtiPhysicalDataT        * phys_data;
  mtiSignalIdT              sigid;
  mtiTypeIdT                typeid;
} signalInfoT;

typedef struct {
  signalInfoT   * sig_info;      /* List of signals. */
  mtiProcessIdT   proc;          /* Test process id. */
} instanceInfoT;

static void printExtraUnits( signalInfoT * siginfo, mtiInt32T value )
{
  char             * unit_name;
  mtiInt32T          num_units;
  mtiInt32T          position;
  mtiInt32T          remainder;
  mtiPhysicalDataT * pdp;

  for ( pdp = siginfo->phys_data; pdp; pdp = pdp->next ) {
   if ( value < pdp->position ) {
    break;
    }
   unit_name = pdp->unit_name;
   position  = pdp->position;
   }
  num_units = value / position;
  remainder = value % position;
  mti_PrintFormatted( " and %d %s", num_units, unit_name );
  if ( remainder ) {
   printExtraUnits( siginfo, remainder );
   }
}

static void checkValues( void *inst_info )
{
  char             * unit_name;
  instanceInfoT    * inst_data = (instanceInfoT *)inst_info;
  mtiInt32T          num_units;
  mtiInt32T          position;
  mtiInt32T          remainder;
  mtiInt32T          sigval;
  mtiPhysicalDataT * pdp;
  signalInfoT      * siginfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
   mti_PrintFormatted( "  Signal %s:", siginfo->name );
   sigval = mti_GetSignalValue( siginfo->sigid );
```

```
      for ( pdp = siginfo->phys_data; pdp; pdp = pdp->next ) {
       if ( sigval < pdp->position ) {
        break;
       }
       unit_name = pdp->unit_name;
       position  = pdp->position;
      }
      num_units = sigval / position;
      remainder = sigval % position;
      mti_PrintFormatted( "  %d = %d %s", sigval, num_units, unit_name );
      if ( remainder ) {
       printExtraUnits( siginfo, remainder );
      }
      mti_PrintFormatted( "\n" );
     }

     mti_ScheduleWakeup( inst_data->proc, 5 );
   }

   static signalInfoT * setupSignal( mtiSignalIdT sigid )
   {
     char              * prev_unit_name;
     mtiInt32T           num_units;
     mtiInt32T           prev_position;
     mtiPhysicalDataT * pdp;
     signalInfoT       * siginfo = 0;

     if ( mti_GetTypeKind( mti_GetSignalType( sigid )) == MTI_TYPE_PHYSICAL ) {
      siginfo            = (signalInfoT *)mti_Malloc(sizeof(signalInfoT));
      siginfo->sigid     = sigid;
      siginfo->name      = mti_GetSignalNameIndirect( sigid, 0, 0 );
      siginfo->typeid    = mti_GetSignalType( sigid );
      siginfo->phys_data = mti_GetPhysicalData( siginfo->typeid );
      siginfo->next      = 0;
      mti_PrintFormatted( "Setting a watch on %s\n", siginfo->name );
      mti_PrintFormatted( "  Physical Units are:\n" );
      for ( pdp = siginfo->phys_data; pdp; pdp = pdp->next ) {
       mti_PrintFormatted( "    %10s = %d %s",
                          pdp->unit_name, pdp->position,
                          siginfo->phys_data->unit_name );
       if ( pdp != siginfo->phys_data ) {
        num_units = pdp->position / prev_position;
        mti_PrintFormatted( " = %d %s", num_units, prev_unit_name );
       }
       mti_PrintFormatted( "\n" );
       prev_unit_name = pdp->unit_name;
       prev_position  = pdp->position;
      }
     }
     return( siginfo );
   }

   static void initInstance( void * param )
   {
     instanceInfoT * inst_data;
     mtiSignalIdT    sigid;
     signalInfoT   * curr_info;
     signalInfoT   * siginfo;
```

```
  inst_data            = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
   siginfo = setupSignal( sigid );
   if ( siginfo ) {
    if ( inst_data->sig_info == 0 ) {
     inst_data->sig_info = siginfo;
    }
    else {
     curr_info->next = siginfo;
    }
    curr_info = siginfo;
   }
  }
  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 4 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bigtime is range 0 to integer'high
   units
     hour;
     day   = 24 hour;
     week  = 7 day;
     month = 4 week;
     year  = 12 month;
   end units;

end top;

architecture a of top is

  signal phys_sig1 : bigtime := 3 day;
  signal phys_sig2 : bigtime := 1 week;
  signal phys_sig3 : bigtime := 1 year;

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  phys_sig1 <= phys_sig1 + 1 day after 5 ns;
  phys_sig2 <= phys_sig2 + 40 hour after 5 ns;
  phys_sig3 <= phys_sig3 + 80 hour after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Setting a watch on phys_sig1
#    Physical Units are:
#            hour = 1 hour
#             day = 24 hour = 24 hour
#            week = 168 hour = 7 day
#           month = 672 hour = 4 week
#            year = 8064 hour = 12 month
# Setting a watch on phys_sig2
#    Physical Units are:
#            hour = 1 hour
#             day = 24 hour = 24 hour
#            week = 168 hour = 7 day
#           month = 672 hour = 4 week
#            year = 8064 hour = 12 month
# Setting a watch on phys_sig3
#    Physical Units are:
#            hour = 1 hour
#             day = 24 hour = 24 hour
#            week = 168 hour = 7 day
#           month = 672 hour = 4 week
#            year = 8064 hour = 12 month
VSIM 1> run 20
# Time [0,4]:
#   Signal phys_sig1:  72 = 3 day
#   Signal phys_sig2:  168 = 1 week
#   Signal phys_sig3:  8064 = 1 year
# Time [0,9]:
#   Signal phys_sig1:  96 = 4 day
#   Signal phys_sig2:  208 = 1 week and 1 day and 16 hour
#   Signal phys_sig3:  8144 = 1 year and 3 day and 8 hour
# Time [0,14]:
#   Signal phys_sig1:  120 = 5 day
#   Signal phys_sig2:  248 = 1 week and 3 day and 8 hour
#   Signal phys_sig3:  8224 = 1 year and 6 day and 16 hour
# Time [0,19]:
#   Signal phys_sig1:  144 = 6 day
#   Signal phys_sig2:  288 = 1 week and 5 day
#   Signal phys_sig3:  8304 = 1 year and 1 week and 3 day
VSIM 2> quit
```

mti_GetPrimaryName()

Gets the primary name of a region (entity, package, or module).

Syntax

primary_name = mti_GetPrimaryName(region_id);

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or System region

Return Values

Name	Type	Description
primary_name	char *	The primary name of the specified region

Description

mti_GetPrimaryName() returns the primary name of the specified VHDL, Verilog, or SystemC region (that is, an entity, package, module, or sc_module name). It does not use the parent primary design unit if the region is not a primary design unit.

You must not free the returned pointer.

Examples

FLI code

```
#include <mti.h>

static void printRegionInfo( char * region )
{
  char *       primary_name;
  char *       region_name;
  mtiRegionIdT regid;

  regid        = mti_FindRegion( region );
  region_name  = mti_GetRegionFullName( regid );
  primary_name = mti_GetPrimaryName( regid );
  mti_PrintFormatted( "  Region %s; Primary name is %s\n",
                      region_name, primary_name );
  mti_VsimFree( region_name );
}

static void initInstance( void * param )
{
  mti_PrintFormatted( "Load Done Callback Function:\n" );
  printRegionInfo( "/top" );
  printRegionInfo( "/top/linst1" );
  printRegionInfo( "/top/linst2" );
  printRegionInfo( "/top/finst" );
  printRegionInfo( "/for_pkg" );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this   */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  char * primary_name;
  char * region_name;

  mti_PrintFormatted( "Foreign Init Function:\n" );
  region_name = mti_GetRegionFullName( region );
  primary_name = mti_GetPrimaryName( region );
  mti_PrintFormatted( "  Region parameter is %s; Primary name is %s\n",
                      region_name, primary_name );
  mti_VsimFree( region_name );
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

package for_pkg is
  procedure test_proc;
end for_pkg;

package body for_pkg is
  procedure test_proc is
   begin
     assert false report "I'm in the test_proc." severity note;
   end;
end for_pkg;

use work.for_pkg.all;

entity lower is
end lower;

architecture level of lower is
begin
  p1 : process
   begin
     test_proc;
     wait for 20 ns;
   end process;
end level;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is
  component for_model
  end component;

  component lower
  end component;

begin
  linst1 : lower;
  linst2 : lower;
  finst  : for_model;
end a;

configuration cfg_top of top is
  for a
   for all : lower
     use entity work.lower(level);
```

```
    end for;
    for all : for_model
     use entity work.for_model(a);
    end for;
   end for;
  end cfg_top;
```

Simulation output

```
% vsim -c cfg_top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c cfg_top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.for_pkg(body)
# Loading work.cfg_top
# Loading work.top(a)
# Loading work.lower(level)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Foreign Init Function:
#    Region parameter is /top/finst; Primary name is for_model
# Load Done Callback Function:
#    Region /top; Primary name is top
#    Region /top/linst1; Primary name is lower
#    Region /top/linst2; Primary name is lower
#    Region /top/finst; Primary name is for_model
#    Region /for_pkg; Primary name is for_pkg
VSIM 1> run 20
# ** Note: I'm in the test_proc.
#    Time: 0 ns  Iteration: 0  Instance: /top/linst2
# ** Note: I'm in the test_proc.
#    Time: 0 ns  Iteration: 0  Instance: /top/linst1
# ** Note: I'm in the test_proc.
#    Time: 20 ns  Iteration: 0  Instance: /top/linst2
# ** Note: I'm in the test_proc.
#    Time: 20 ns  Iteration: 0  Instance: /top/linst1
VSIM 2> quit
```

mti_GetProcessName()

Gets the name of a process.

Syntax

proc_name = mti_GetProcessName(proc_id)

Arguments

Name	Type	Description
proc_id	mtiProcessIdT	A handle to a VHDL or SystemC process

Return Values

Name	Type	Description
proc_name	char *	The name of the specified process

Description

mti_GetProcessName() returns the name of the specified process. You must not free the returned pointer.

Examples

FLI code

```
#include <mti.h>

void printProcesses( mtiRegionIdT region, int indent )
{
  mtiProcessIdT procid;

  for ( procid = mti_FirstProcess( region ); procid;
        procid = mti_NextProcess() ) {
   if ( procid ) {
    mti_PrintFormatted( "%*cProcess %s\n", indent, ' ',
                        mti_GetProcessName( procid ) );
   }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printProcesses( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
   printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;

  p1 : process
   variable count : integer := 0;
  begin
   count := count + 1;
   wait on a;
  end process;
end b;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
   generic ( delay : time := 5 ns );
   port ( a : in bit;
          b : out bit
        );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
```

```
    s3 <= not s3 after 5 ns;

    toggle : inv port map ( s1, s2 );

    proc1 : process
     variable count : integer := 0;
    begin
     wait on s1;
     count := count + 1;
    end process proc1;

  end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading work.top(a)
    # Loading work.inv(b)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    #
    # Elaboration phase:
    #  Region /top
    #    Region /top/i1
    #    Region /top/flip
    #       Process p1
    #       Process line__19
    #
    # Load Done phase:
    #  Region /top
    #    Process proc1
    #    Process line__58
    #    Process line__57
    #    Region /top/flip
    #       Process p1
    #       Process line__19
    #    Region /top/i1
    #    Region /top/toggle
    #       Process p1
    #       Process line__19
    VSIM 1> run 10
    VSIM 2> quit
```

mti_GetProcessRegion()

Returns a scope to a process' region.

Syntax

region = mti_GetProcessRegion(proc_id)

Arguments

Name	Type	Description
proc_id	mtiProcessIdT	A handle to a process

Return Values

Name	Type	Description
region	mtiRegionIdT	A handle to the region in which the process exists

Description

mti_GetProcessRegion returns the VHDL or SystemC scope of the last line to execute in the specified process.

Examples

FLI code

```c
#include <mti.h>

void printProcesses( mtiRegionIdT region, int indent )
{
    char *          region_name;
    mtiProcessIdT procid;
    mtiRegionIdT  regid;

    for ( procid = mti_FirstProcess( region ); procid;
          procid = mti_NextProcess() ) {
        if ( procid ) {
            regid = mti_GetProcessRegion(procid);
            region_name = mti_GetRegionFullName( regid );
            mti_PrintFormatted( "%*cProcess %s is in region %s\n",
                                indent, ' ',
                                mti_GetProcessName( procid ),
                                region_name
                                );
            mti_VsimFree( region_name );
        }
    }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
    mtiRegionIdT regid;

    printProcesses( region, indent );
    for ( regid = mti_FirstLowerRegion( region );
          regid; regid = mti_NextRegion( regid ) ) {
        printHierarchy( regid, indent );
    }
}

void loadDoneCB( void * param )
{
    mti_PrintMessage( "\nLoad Done phase:\n" );
    printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
    mtiRegionIdT       region,   /* The ID of the region in which this      */
                                 /* foreign architecture is instantiated.   */
    char              *param,    /* The last part of the string in the      */
                                 /* foreign attribute.                      */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
    mti_AddLoadDoneCB( loadDoneCB, 0 );
    mti_PrintMessage( "\nElaboration phase:\n" );
    printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;

  p1 : process
    variable count : integer := 0;
  begin
    count := count + 1;
    wait on a;
  end process;
end b;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
```

```
    s3 <= not s3 after 5 ns;

    toggle : inv port map ( s1, s2 );

    proc1 : process
      variable count : integer := 0;
    begin
      wait on s1;
      count := count + 1;
    end process proc1;

  end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.6

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading work.top(a)
    # Loading work.inv(b)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    #
    # Elaboration phase:
    #  Process p1 is in region /top/flip
    #  Process line__19 is in region /top/flip
    #
    # Load Done phase:
    #  Process proc1 is in region /top
    #  Process line__58 is in region /top
    #  Process line__57 is in region /top
    #  Process p1 is in region /top/flip
    #  Process line__19 is in region /top/flip
    #  Process p1 is in region /top/toggle
    #  Process line__19 is in region /top/toggle
    VSIM 1> run 10
    VSIM 2> quit
```

mti_GetProductVersion()

Gets the name and version of the simulator.

Syntax

prod_ver = mti_GetProductVersion()

Arguments

None

Return Values

Name	Type	Description
prod_ver	char *	The name and version of the product

Description

mti_GetProductVersion() returns the name and version of the product. You must not free the returned pointer.

Examples

FLI code

```
#include <mti.h>

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
mti_PrintFormatted( "The version of the simulator is:\n  \"%s\".\n",
                   mti_GetProductVersion() );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# The version of the simulator is:
#   "<Product Name> Version 6.2a 2006.6".
VSIM 1> quit
```

mti_GetRegionFullName()

Gets the full hierarchical name of a region.

Syntax

region_name = mti_GetRegionFullName(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
region_name	char *	The full hierarchical name of the specified region

Description

mti_GetRegionFullName() returns the full hierarchical name of the specified VHDL, Verilog, or SystemC region. The caller is responsible for freeing the returned pointer with mti_VsimFree().

Examples

FLI code

```c
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
   printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char               *param,   /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
```

```
end top;
architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#  Region /top
#     Region /top/inst1
#        Region /top/inst1/i1
#        Region /top/inst1/flip
#
# Load Done phase:
#  Region /top
#     Region /top/inst1
#        Region /top/inst1/flip
#        Region /top/inst1/i1
#        Region /top/inst1/toggle
VSIM 1> quit
```

mti_GetRegionKind()

Gets the type of a region (VHDL, Verilog, or SystemC).

Syntax

region_kind = mti_GetRegionKind(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
region_kind	int	The kind of the region

Description

mti_GetRegionKind() returns the kind of the specified VHDL, Verilog, or SystemC region. The value returned is one of the type (not fulltype) values defined in *acc_user.h* or *acc_vhdl.h*. You can use the PLI routine acc_fetch_fulltype() on the region_id to get the fulltype of the region. If the region_id is a handle to a Verilog region, then you can use it with PLI functions to obtain information about and access objects in the Verilog region.

Examples

FLI code

```
#include <acc_user.h>
#include <acc_vhdl.h>
#include <mti.h>

static void printFullType( handle region )
{
  int fulltype = acc_fetch_fulltype( region );

  switch ( fulltype ) {
    case accArchitecture:
      mti_PrintFormatted( " of fulltype accArchitecture" );
      break;
    case accArchVitalLevel0:
      mti_PrintFormatted( " of fulltype accArchVitalLevel0" );
      break;
    case accArchVitalLevel1:
      mti_PrintFormatted( " of fulltype accArchVitalLevel1" );
      break;
    case accEntityVitalLevel0:
      mti_PrintFormatted( " of fulltype accEntityVitalLevel0" );
      break;
    case accForeignArch:
      mti_PrintFormatted( " of fulltype accForeignArch" );
      break;
    case accForeignArchMixed:
      mti_PrintFormatted( " of fulltype accForeignArchMixed" );
      break;
    case accFunction:
      mti_PrintFormatted( " of fulltype accFunction" );
      break;
    case accModuleInstance:
      mti_PrintFormatted( " of fulltype accModuleInstance" );
      break;
    case accPackage:
      mti_PrintFormatted( " of fulltype accPackage" );
      break;
    case accShadow:
      mti_PrintFormatted( " of fulltype accShadow" );
      break;
    case accTask:
      mti_PrintFormatted( " of fulltype accTask" );
      break;
    default:
      mti_PrintFormatted( " of fulltype %d", fulltype );
      break;
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
```

```
        mti_PrintFormatted( "%*cRegion %s is ", indent, ' ', region_name );
        switch ( mti_GetRegionKind( region ) ) {
          case accArchitecture:
            mti_PrintFormatted( "a VHDL architecture" );
            printFullType( region );
            break;
          case accForeign:
            mti_PrintFormatted( "an FLI-created region" );
            printFullType( region );
            break;
          case accFunction:
            mti_PrintFormatted( "a Verilog function" );
            printFullType( region );
            break;
          case accModule:
            mti_PrintFormatted( "a Verilog module" );
            printFullType( region );
            break;
          case accPackage:
            mti_PrintFormatted( "a VHDL package" );
            printFullType( region );
            break;
          case accTask:
            mti_PrintFormatted( "a Verilog task" );
            printFullType( region );
            break;
          default:
            mti_PrintFormatted( "UNKNOWN" );
            printFullType( region );
            break;
        }
        mti_PrintFormatted( "\n" );
        indent += 2;
        for ( regid = mti_FirstLowerRegion( region );
              regid; regid = mti_NextRegion( regid ) ) {
          printHierarchy( regid, indent );
        }
        mti_VsimFree( region_name );
      }

      void loadDoneCB( void * param )
      {
        mtiRegionIdT regid;

        mti_PrintMessage( "\nDesign Regions:\n" );
        for ( regid = mti_GetTopRegion(); regid; regid = mti_NextRegion(regid) ) {
          printHierarchy( regid, 1 );
        }
      }
      void initForeign(
        mtiRegionIdT        region,    /* The ID of the region in which this    */
                                       /* foreign architecture is instantiated. */
        char               *param,     /* The last part of the string in the    */
                                       /* foreign attribute.                    */
        mtiInterfaceListT *generics,   /* A list of generics for the foreign model.*/
        mtiInterfaceListT *ports       /* A list of ports for the foreign model.   */
      )
      {
```

```
    (void) mti_CreateRegion( region, "my_region" );
    mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
cache.v
module cache(clk, paddr, pdata, prw, pstrb, prdy,
             saddr, sdata, srw, sstrb, srdy);
  input  clk, srdy, paddr, prw, pstrb;
  output      prdy, saddr, srw, sstrb;
  inout  sdata, pdata;

  `define addr_size  8
  `define set_size   5
  `define word_size  16

  reg verbose;

  reg [`word_size-1:0] sdata_r, pdata_r;
  reg [`addr_size-1:0] saddr_r;
  reg                  srw_r, sstrb_r, prdy_r;

  wire [`addr_size-1:0]      paddr;
  wire [`addr_size-1:0] #(5) saddr = saddr_r;
  wire [`word_size-1:0] #(5) sdata = sdata_r, pdata = pdata_r;
  wire                  #(5) srw   = srw_r, sstrb = sstrb_r, prdy = prdy_r;

  reg  [3:0] oen, wen;
  wire [3:0] hit;

  /*************** Cache sets ***************/
  cache_set s0(paddr, pdata, hit[0], oen[0], wen[0]);
  cache_set s1(paddr, pdata, hit[1], oen[1], wen[1]);
  cache_set s2(paddr, pdata, hit[2], oen[2], wen[2]);
  cache_set s3(paddr, pdata, hit[3], oen[3], wen[3]);

  initial begin
    verbose = 1;
    saddr_r = 0;
    sdata_r = 'bz;
    pdata_r = 'bz;
    srw_r = 0;
    sstrb_r = 1;
    prdy_r = 1;
    oen = 4'b1111;
    wen = 4'b1111;
  end

  /*************** Local MRU memory ***************/

  reg [2:0] mru_mem [0:(1 << `set_size) - 1];

  integer i;
  initial for (i = 0; i < (1 << `set_size); i=i+1) mru_mem[i] = 0;

  function integer hash;
    input [`addr_size-1:0] a;
    hash = a[`set_size - 1:0];
  endfunction

  task update_mru;
```

```
      input [`addr_size-1:0] addr;
      input [3:0] hit;
      reg [2:0] mru;
      begin
        mru = mru_mem[hash(addr)];
        mru[2] = ((hit & 4'b1100) != 0);
        if (mru[2]) mru[1] = hit[3];
        else        mru[0] = hit[1];
        mru_mem[hash(addr)] = mru;
      end
  endtask

  function [3:0] pick_set;
      input [`addr_size-1:0] addr;
      integer setnum;
      begin
        casez (mru_mem[hash(addr)])
          3'b1?1 : setnum = 0;
          3'b1?0 : setnum = 1;
          3'b01? : setnum = 2;
          3'b00? : setnum = 3;
          default: setnum = 0;
        endcase
        if (verbose) begin
          if (prw == 1)
            $display("%t: Read miss, picking set %0d", $time, setnum);
          else
            $display("%t: Write miss, picking set %0d", $time, setnum);
        end
        pick_set = 4'b0001 << setnum;
      end
  endfunction

  /*************** System Bus interface ***************/
  task sysread;
      input  [`addr_size-1:0] a;
      begin
        saddr_r = a;
        srw_r = 1;
        sstrb_r = 0;
        @(posedge clk) sstrb_r = 1;
        assign prdy_r = srdy;
        assign pdata_r = sdata;
        @(posedge clk) while (srdy != 0) @(posedge clk) ;
        deassign prdy_r;  prdy_r = 1;
        deassign pdata_r; pdata_r = 'bz;
      end
  endtask

  task syswrite;
      input  [`addr_size-1:0] a;
      begin
        saddr_r = a;
        srw_r = 0;
        sstrb_r = 0;
        @(posedge clk) sstrb_r = 1;
        assign prdy_r = srdy;
        assign sdata_r = pdata;
```

```verilog
      @(posedge clk) while (srdy != 0) @(posedge clk) ;
      deassign prdy_r;  prdy_r = 1;
      deassign sdata_r; sdata_r = 'bz;
      sdata_r = 'bz;
    end
  endtask

  /*************** Cache control ***************/

  function [3:0] get_hit;
    input [3:0] hit;
    integer setnum;
    begin
      casez (hit)
        4'b???1 : setnum = 0;
        4'b??1? : setnum = 1;
        4'b?1?? : setnum = 2;
        4'b1??? : setnum = 3;
      endcase
      if (verbose) begin
        if (prw == 1)
          $display("%t: Read hit to set %0d", $time, setnum);
        else
          $display("%t: Write hit to set %0d", $time, setnum);
      end
      get_hit = 4'b0001 << setnum;
    end
  endfunction

  reg [3:0] setsel;
  always @(posedge clk) if (pstrb == 0) begin
    if ((prw == 1) && hit) begin
      // Read Hit..
      setsel = get_hit(hit);
      oen = ~setsel;
      prdy_r = 0;
      @(posedge clk) prdy_r = 1;
      oen = 4'b1111;
    end else begin
      // Read Miss or Write Hit..
      if (hit)
        setsel = get_hit(hit);
      else
        setsel = pick_set(paddr);
      wen = ~setsel;
      if (prw == 1)
        sysread (paddr);
      else
        syswrite(paddr);
      wen = 4'b1111;
    end
    update_mru(paddr, setsel);
  end
endmodule
memory.v
module memory(clk, addr, data, rw, strb, rdy);
  input  clk, addr, rw, strb;
  output rdy;
```

```
  inout   data;

  `define addr_size 8
  `define word_size 16

  reg [`word_size-1:0] data_r;
  reg                  rdy_r;

  initial begin
    data_r = 'bz;
    rdy_r = 1;
  end

  wire [`addr_size-1:0] addr;
  wire [`word_size-1:0] #(5) data = data_r;
  wire                  #(5) rdy = rdy_r;

  reg [`word_size-1:0] mem[0:(1 << `addr_size) - 1];

  integer i;
  always @(posedge clk) if (strb == 0) begin
    i = addr;
    repeat (2) @(posedge clk) ;
    if (rw == 1)
      data_r = mem[i];
    rdy_r = 0;
    @(posedge clk)
    rdy_r = 1;
    if (rw == 0)
      mem[i] = data;
    else
      data_r = 'bz;
  end
endmodule
proc.v
module proc(clk, addr, data, rw, strb, rdy);
  input   clk, rdy;
  output addr, rw, strb;
  inout   data;

  `define addr_size 8
  `define word_size 16

  reg [`addr_size-1:0] addr_r;
  reg [`word_size-1:0] data_r;
  reg                  rw_r, strb_r;

  reg verbose;

  wire [`addr_size-1:0] #(5) addr = addr_r;
  wire [`word_size-1:0] #(5) data = data_r;
  wire                  #(5) rw = rw_r, strb = strb_r;

  task read;
    input  [`addr_size-1:0] a;
    output [`word_size-1:0] d;
    begin
      if (verbose) $display("%t: Reading from addr=%h", $time, a);
```

```
        addr_r = a;
        rw_r = 1;
        strb_r = 0;
        @(posedge clk) strb_r = 1;
        @(posedge clk) while (rdy != 0) @(posedge clk) ;
        d = data;
      end
  endtask

  task write;
    input  [`addr_size-1:0] a;
    input  [`word_size-1:0] d;
    begin
      if (verbose)
       $display("%t: Writing data=%h to addr=%h", $time, d, a);
      addr_r = a;
      rw_r = 0;
      strb_r = 0;
      @(posedge clk) strb_r = 1;
      data_r = d;
      @(posedge clk) while (rdy != 0) @(posedge clk) ;
      data_r = 'bz;
    end
  endtask

  reg [`addr_size-1:0] a;
  reg [`word_size-1:0] d;
  initial begin
    // Set initial state of outputs..
    addr_r = 0;
    data_r = 'bz;
    rw_r = 0;
    strb_r = 1;
    verbose = 1;

    forever begin
      // Wait for first clock, then perform read/write test
      @(posedge clk)
      if (verbose) $display("%t: Starting Read/Write test", $time);

      // Write 10 locations
      for (a = 0; a < 10; a = a + 1)
        write(a, a);

      // Read back 10 locations
      for (a = 0; a < 10; a = a + 1) begin
        read(a, d);
        if (d !== a)
          $display("%t: Read/Write mismatch; E: %h, A: %h", $time, a, d);
      end

      if (verbose) $display("Read/Write test done");
      $stop(1);
    end
  end
endmodule
util.vhd
library IEEE;
```

```
use IEEE.std_logic_1164.all;

package std_logic_util is
  function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
                                        return STD_LOGIC_VECTOR;
  function CONV_INTEGER(ARG: STD_LOGIC_VECTOR) return INTEGER;
end std_logic_util;

package body std_logic_util is
  type tbl_type is array (STD_ULOGIC) of STD_ULOGIC;
  constant tbl_BINARY : tbl_type :=
    ('0', '0', '0', '1', '0', '0', '0', '1', '0');

  function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
                                        return STD_LOGIC_VECTOR is
    variable result: STD_LOGIC_VECTOR(SIZE-1 downto 0);
    variable temp: integer;
  begin
    temp := ARG;
    for i in 0 to SIZE-1 loop
      if (temp mod 2) = 1 then
        result(i) := '1';
      else
        result(i) := '0';
      end if;
      if temp > 0 then
        temp := temp / 2;
      else
        temp := (temp - 1) / 2; -- simulate ASR
      end if;
    end loop;
    return result;
  end;

  function CONV_INTEGER(ARG: STD_LOGIC_VECTOR) return INTEGER is
    variable result: INTEGER;
  begin
    assert ARG'length <= 32
      report "ARG is too large in CONV_INTEGER"
      severity FAILURE;
    result := 0;
    for i in ARG'range loop
      if i /= ARG'left then
        result := result * 2;
        if tbl_BINARY(ARG(i)) = '1' then
          result := result + 1;
        end if;
      end if;
    end loop;
    return result;
  end;
end std_logic_util;
set.vhd
library ieee;
use ieee.std_logic_1164.all;
use work.std_logic_util.all;

entity cache_set is
```

```
generic(
  addr_size  : integer := 8;
  set_size   : integer := 5;
  word_size  : integer := 16
);
port(
  addr           : in    std_logic_vector(addr_size-1 downto 0);
  data           : inout std_logic_vector(word_size-1 downto 0);
  hit            : out   std_logic;
  oen            : in    std_logic;
  wen            : in    std_logic
);
end cache_set;

architecture only of cache_set is
  constant size : integer := 2**set_size;
  constant dly : time := 5 ns;
  subtype word_t is std_logic_vector(word_size-1 downto 0);
  subtype addr_t is std_logic_vector(addr_size-1 downto 0);
  type mem_t is array (0 to size-1) of word_t;
  subtype tag_word_t is std_logic_vector(addr_size-1 downto set_size);
  type tag_t is array (0 to size-1) of tag_word_t;
  type valid_t is array (0 to size-1) of boolean;
  signal data_out : word_t;
begin

  data <= (others => 'Z') after dly when (oen = '1') else data_out after
dly;

process(wen, addr)
    ---------- Local tag and data memories -----------
    variable data_mem : mem_t;
    variable atag_mem : tag_t;
    variable valid_mem : valid_t := (others => false);

    function hash(constant a : addr_t) return integer is
    begin
      return conv_integer(a(set_size-1 downto 0));
    end;

    procedure lookup_cache(constant a : addr_t) is
      variable i : integer;
      variable found : boolean;
    begin
      i := hash(a);
      found := valid_mem(i) and (a(tag_word_t'range) = atag_mem(i));
      if found then
        hit <= '1' after dly;
      else
        hit <= '0' after dly;
      end if;
    end;

    procedure update_cache(constant a : addr_t;
                           constant d : word_t) is
      variable i : integer;
    begin
      i := hash(a);
```

```
        data_mem(i) := d;
        atag_mem(i) := a(tag_word_t'range);
        valid_mem(i) := true;
      end;

  begin
    if wen'event and (wen = '1') then
      update_cache(addr, data);
    end if;

    lookup_cache(addr);

    data_out <= data_mem(hash(addr));
  end process;
end;
top.vhd
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is end;

architecture only of top is
  component proc
    port(
      clk              : in    std_logic;
      addr             : out   std_logic_vector(7 downto 0);
      data             : inout std_logic_vector(15 downto 0);
      rw               : out   std_logic;
      strb             : out   std_logic;
      rdy              : in    std_logic
    );
  end component;

  component cache
    port(
      clk              : in    std_logic;
      paddr            : in    std_logic_vector(7 downto 0);
      pdata            : inout std_logic_vector(15 downto 0);
      prw              : in    std_logic;
      pstrb            : in    std_logic;
      prdy             : out   std_logic;
      saddr            : out   std_logic_vector(7 downto 0);
      sdata            : inout std_logic_vector(15 downto 0);
      srw              : out   std_logic;
      sstrb            : out   std_logic;
      srdy             : in    std_logic
    );
  end component;

  component memory
```

```
    port(
      clk              : in    std_logic;
      addr             : in    std_logic_vector(7 downto 0);
      data             : inout std_logic_vector(15 downto 0);
      rw               : in    std_logic;
      strb             : in    std_logic;
      rdy              : out   std_logic
    );
  end component;

  component for_model
  end component;

  signal clk : std_logic := '0';

  -- Processor bus signals
  signal prw, pstrb, prdy : std_logic;
  signal paddr : std_logic_vector(7 downto 0);
  signal pdata : std_logic_vector(15 downto 0);

  -- System bus signals
  signal srw, sstrb, srdy : std_logic;
  signal saddr : std_logic_vector(7 downto 0);
  signal sdata : std_logic_vector(15 downto 0);
begin
  clk <= not clk after 20 ns;

  p: proc   port map(clk, paddr, pdata, prw, pstrb, prdy);

  c: cache  port map(clk, paddr, pdata, prw, pstrb, prdy,
                                    saddr, sdata, srw, sstrb, srdy);

  m: memory port map(clk, saddr, sdata, srw, sstrb, srdy);

  inst1 : for_model;
end;
```

Simulation output

```
% vlog cache.v memory.v proc.v
Model Technology ModelSim SE/EE vlog 5.4b Compiler 2000.06 Jun  9 2000
-- Compiling module cache
-- Compiling module memory
-- Compiling module proc

Top level modules:
    cache
    memory
    proc
% vcom util.vhd set.vhd
Model Technology ModelSim SE/EE vcom 5.4b Compiler 2000.06 Jun  9 2000
-- Loading package standard
-- Loading package std_logic_1164
-- Compiling package std_logic_util
-- Compiling package body std_logic_util
-- Loading package std_logic_util
-- Loading package std_logic_util
-- Compiling entity cache_set
-- Compiling architecture only of cache_set
% vcom -93 top.vhd
Model Technology ModelSim SE/EE vcom 5.4b Compiler 2000.06 Jun  9 2000
-- Loading package standard
-- Compiling entity for_model
-- Compiling architecture a of for_model
-- Loading package std_logic_1164
-- Compiling entity top
-- Compiling architecture only of top
-- Loading package vl_types
-- Loading entity proc
-- Loading entity cache
-- Loading entity memory
-- Loading entity for_model
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading .../modeltech/sunos5/../verilog.vl_types(body)
# Loading work.top(only)
# Loading work.proc
# Loading work.cache
# Loading work.std_logic_util(body)
# Loading work.cache_set(only)
# Loading work.memory
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Design Regions:
#  Region /top is a VHDL architecture of fulltype accArchitecture
#     Region /top/p is a Verilog module of fulltype accModuleInstance
#        Region /top/p/read is a Verilog task of fulltype accTask
#        Region /top/p/write is a Verilog task of fulltype accTask
```

```
#     Region /top/c is a Verilog module of fulltype accModuleInstance
#       Region /top/c/hash is a Verilog function of fulltype accFunction
#       Region /top/c/update_mru is a Verilog task of fulltype accTask
#       Region /top/c/pick_set is a Verilog function of fulltype
accFunction
#       Region /top/c/sysread is a Verilog task of fulltype accTask
#       Region /top/c/syswrite is a Verilog task of fulltype accTask
#       Region /top/c/get_hit is a Verilog function of fulltype accFunction
#       Region /top/c/s0 is a VHDL architecture of fulltype accArchitecture
#       Region /top/c/s1 is a VHDL architecture of fulltype accArchitecture
#       Region /top/c/s2 is a VHDL architecture of fulltype accArchitecture
#       Region /top/c/s3 is a VHDL architecture of fulltype accArchitecture
#     Region /top/m is a Verilog module of fulltype accModuleInstance
#     Region /top/inst1 is a VHDL architecture of fulltype accForeignArch
#       Region /top/inst1/my_region is an FLI-created region of fulltype
accShadow
#   Region /standard is a VHDL package of fulltype accPackage
#   Region /std_logic_1164 is a VHDL package of fulltype accPackage
#   Region /vl_types is a VHDL package of fulltype accPackage
#   Region /std_logic_util is a VHDL package of fulltype accPackage
VSIM 1> run -all
#                   20: Starting Read/Write test
#                   20: Writing data=0000 to addr=00
#                   60: Write miss, picking set 3
#                  220: Writing data=0001 to addr=01
#                  260: Write miss, picking set 3
#                  420: Writing data=0002 to addr=02
#                  460: Write miss, picking set 3
#                  620: Writing data=0003 to addr=03
#                  660: Write miss, picking set 3
#                  820: Writing data=0004 to addr=04
#                  860: Write miss, picking set 3
#                 1020: Writing data=0005 to addr=05
#                 1060: Write miss, picking set 3
#                 1220: Writing data=0006 to addr=06
#                 1260: Write miss, picking set 3
#                 1420: Writing data=0007 to addr=07
#                 1460: Write miss, picking set 3
#                 1620: Writing data=0008 to addr=08
#                 1660: Write miss, picking set 3
#                 1820: Writing data=0009 to addr=09
#                 1860: Write miss, picking set 3
#                 2020: Reading from addr=00
#                 2060: Read hit to set 3
#                 2100: Reading from addr=01
#                 2140: Read hit to set 3
#                 2180: Reading from addr=02
#                 2220: Read hit to set 3
#                 2260: Reading from addr=03
#                 2300: Read hit to set 3
#                 2340: Reading from addr=04
#                 2380: Read hit to set 3
#                 2420: Reading from addr=05
#                 2460: Read hit to set 3
#                 2500: Reading from addr=06
#                 2540: Read hit to set 3
#                 2580: Reading from addr=07
#                 2620: Read hit to set 3
```

```
#                       2660: Reading from addr=08
#                       2700: Read hit to set 3
#                       2740: Reading from addr=09
#                       2780: Read hit to set 3
# Read/Write test done
# ** Note: $stop    : proc.v(77)
#    Time: 2820 ns  Iteration: 0  Instance: /top/p
# Break at proc.v line 77
# Stopped at proc.v line 77
VSIM 2> quit
```

mti_GetRegionName()

Gets the simple name of a region.

Syntax

region_name = mti_GetRegionName(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
region_name	char *	The simple name of the specified region

Description

mti_GetRegionName() returns the simple name of the specified VHDL, Verilog, or SystemC region. You must not free the returned pointer.

Examples

FLI code

```c
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
```

```
        end top;

        architecture a of top is
          component mid is
          end component;
        begin
          inst1 : mid;
        end a;
```

Simulation output

```
        % vsim -c top
        Reading .../modeltech/tcl/vsim/pref.tcl

        # 5.4b

        # vsim -c top
        # Loading .../modeltech/sunos5/../std.standard
        # Loading work.top(a)
        # Loading work.mid(a)
        # Loading work.inv(b)
        # Loading work.for_model(a)
        # Loading ./for_model.sl
        #
        # Load Done phase:
        #   Region top
        #     Region inst1
        #       Region flip
        #       Region i1
        #       Region toggle
        VSIM 1> quit
```

mti_GetRegionSourceName()

Gets the name of the source file which contains a region.

Syntax

source_name = mti_GetRegionSourceName(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
source_name	char *	The name of the source file which contains the specified region

Description

mti_GetRegionSourceName() returns the name of the source file which contains the specified VHDL, Verilog, or SystemC region. The returned pointer must not be freed.

Examples

FLI code

```c
#include "mti.h"

static void printRegionInfo( mtiRegionIdT regid )
{
  char * source_name;
  char * region_name;

  if ( regid ) {
    region_name = mti_GetRegionFullName( regid );
    source_name = mti_GetRegionSourceName( regid );
    mti_PrintFormatted( "  Region %s is in File %s\n",
                        region_name, source_name );
    mti_VsimFree( region_name );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printRegionInfo( mti_FindRegion( "top" ) );
  printRegionInfo( mti_FindRegion( "inst1" ) );
  printRegionInfo( mti_FindRegion( "inst1/i1" ) );
  printRegionInfo( mti_FindRegion( "inst1/flip" ) );
  printRegionInfo( mti_FindRegion( "inst1/toggle" ) );
}
void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  char * source_name;

  mti_AddLoadDoneCB( loadDoneCB, 0 );

  mti_PrintMessage( "\nElaboration phase:\n" );
  source_name = mti_GetRegionSourceName( region );
  mti_PrintFormatted( "  Foreign architecture region is in File %s\n",
                      source_name );
}
```

HDL code

```
for_model.vhd
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;
inv.vhd
entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;
mid.vhd
entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;
  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;
  for all : inv use entity work.inv(b);

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;
top.vhd
entity top is
```

```
end top;

architecture a of top is
  component mid is
  end component;
  for all : mid use entity work.mid(a);
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vcom -93 for_model.vhd inv.vhd mid.vhd top.vhd
Model Technology ModelSim SE/EE vcom 5.4b Compiler 2000.06 Jun  9 2000
-- Loading package standard
-- Compiling entity for_model
-- Compiling architecture a of for_model
-- Compiling entity inv
-- Compiling architecture b of inv
-- Compiling entity mid
-- Compiling architecture a of mid
-- Loading entity for_model
-- Loading entity inv
-- Compiling entity top
-- Compiling architecture a of top
-- Loading entity mid
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#    Foreign architecture region is in File for_model.vhd
#
# Load Done phase:
#    Region /top is in File top.vhd
#    Region /top/inst1 is in File mid.vhd
#    Region /top/inst1/i1 is in File for_model.vhd
#    Region /top/inst1/flip is in File inv.vhd
#    Region /top/inst1/toggle is in File inv.vhd
VSIM 1> quit
```

mti_GetResolutionLimit()

Gets the simulator resolution limit.

Syntax

limit = mti_GetResolutionLimit()

Arguments

None

Return Values

Name	Type	Description
limit	int	The simulator resolution limit in log10 seconds

Description

mti_GetResolutionLimit() returns the simulator resolution limit in log10 seconds. In other words, mti_GetResolutionLimit() returns n from the expression:

```
time_scale = 1*10^n seconds
```

The values returned by mti_GetResolutionLimit() are as follows:

limit	time_scale
2	100 sec
1	10 sec
0	1 sec
-1	100 ms
-2	10 ms
-3	1 ms
-4	100 us
-5	10 us
-6	1 us
-7	100 ns
-8	10 ns
-9	1 ns
-10	100 ps
-11	10 ps

limit	time_scale
-12	1 ps
-13	100 fs
-14	10 fs
-15	1 fs

Examples

FLI code

```
#include <mti.h>

static char * convertLimit( int limit )
{
  switch ( limit ) {
    case 2:    return( "100 sec" );
    case 1:    return( "10 sec" );
    case 0:    return( "1 sec" );
    case -1:   return( "100 ms" );
    case -2:   return( "10 ms" );
    case -3:   return( "1 ms" );
    case -4:   return( "100 us" );
    case -5:   return( "10 us" );
    case -6:   return( "1 us" );
    case -7:   return( "100 ns" );
    case -8:   return( "10 ns" );
    case -9:   return( "1 ns" );
    case -10:  return( "100 ps" );
    case -11:  return( "10 ps" );
    case -12:  return( "1 ps" );
    case -13:  return( "100 fs" );
    case -14:  return( "10 fs" );
    case -15:  return( "1 fs" );
    default:   return( "Unexpected limit" );
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_PrintFormatted( "The resolution limit of the simulator is \"%s\".\n",
                   convertLimit( mti_GetResolutionLimit() ) );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is
end top;

architecture a of top is

   signal s1 : bit := '0';

   component for_model is
   end component;

   for all : for_model use entity work.for_model(a);

begin

   i1 : for_model;

   s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c -t 10ps top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c -t 10ps top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# The resolution limit of the simulator is "10 ps".
VSIM 1> quit
% vsim -c -t 100fs top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c -t 100fs top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# The resolution limit of the simulator is "100 fs".
VSIM 1> quit
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# The resolution limit of the simulator is "1 ns".
VSIM 1> quit
```

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mti_GetRestoreDirname()

Returns the name of the directory of the restore command.

Syntax

dirname = mti_GetRestoreDirname()

Arguments

None

Return Values

Name	Type	Description
dirname	char *	Directory name you specified for the restore command.

Description

This call returns NULL if you specfied a filename for the restore command.

mti_GetRunStopTime()

Gets the stop time of the current simulation run.

Syntax

mti_GetRunStopTime(stop_time)

Arguments

Name	Type	Description
stop_time	mtiTime64T *	Returns the stop time of the current simulation run

Return Values

Nothing

Description

mti_GetRunStopTime() returns the stop time of the current simulation run in the stop_time parameter.

Examples

FLI code

```
#include <mti.h>

static void checkStopTime( void * param )
{
  mtiTime64T stop_time;

  mti_GetRunStopTime( &stop_time );
  mti_PrintFormatted( "Time [%d,%d]: Run stop time is [%d,%d]\n",
                      mti_NowUpper(), mti_Now(),
                      MTI_TIME64_HI32(stop_time),
                      MTI_TIME64_LO32(stop_time) );
}

static void initInstance( void * param )
{
  mtiProcessIdT procid;

  procid = mti_CreateProcess( "Test Process", checkStopTime, 0 );
  mti_Sensitize( procid, mti_FindSignal( "/top/s1" ), MTI_EVENT );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

entity top is
end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

  signal s1 : bit := '0';

begin

  s1 <= not s1 after 5 ns;

  finst  : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 6
# Time [0,5]: Run stop time is [0,6]
VSIM 2> run 7
# Time [0,10]: Run stop time is [0,13]
VSIM 3> run 6
# Time [0,15]: Run stop time is [0,19]
VSIM 4> quit
```

mti_GetSecondaryName()

Gets the secondary name of a VHDL region.

Syntax

sec_name = mti_GetSecondaryName(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL region

Return Values

Name	Type	Description
sec_name	char *	The secondary name of the specified region

Description

mti_GetSecondaryName() returns the secondary name of the specified region; that is, an architecture name. If the region is not a secondary design unit, then the parent secondary design unit is used. A NULL is returned if the region is a VHDL package.

You must not free The returned pointer.

Examples

FLI code

```
#include <mti.h>

static void printRegionInfo( char * region )
{
  char *        primary_name;
  char *         region_name;
  char *         secondary_name;
  mtiRegionIdT regid;

  regid          = mti_FindRegion( region );
  region_name    = mti_GetRegionFullName( regid );
  primary_name   = mti_GetPrimaryName( regid );
  secondary_name = mti_GetSecondaryName( regid );
  mti_PrintFormatted( "  Region %s; Primary name is %s, "
                       "Secondary name is %s\n",
                        region_name, primary_name,
                        secondary_name ? secondary_name : "<NULL>" );
  mti_VsimFree( region_name );
}

static void initInstance( void * param )
{
  mti_PrintFormatted( "Load Done Callback Function:\n" );

  printRegionInfo( "/top" );
  printRegionInfo( "/top/linst1" );
  printRegionInfo( "/top/linst2" );
  printRegionInfo( "/top/finst" );
  printRegionInfo( "/for_pkg" );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  char * primary_name;
  char * region_name;
  char * secondary_name;

  mti_PrintFormatted( "Foreign Init Function:\n" );

  region_name    = mti_GetRegionFullName( region );
  primary_name   = mti_GetPrimaryName( region );
  secondary_name = mti_GetSecondaryName( region );
  mti_PrintFormatted( "  Region parameter is %s; Primary name is %s, "
                       "Secondary name is %s\n",
                        region_name, primary_name,
                        secondary_name ? secondary_name : "<NULL>" );
  mti_VsimFree( region_name );
```

```
    mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture for_arch of for_model is
 attribute foreign of for_arch : architecture is "initForeign
for_model.sl;";
begin
end for_arch;

package for_pkg is

 procedure test_proc;

end for_pkg;

package body for_pkg is

 procedure test_proc is
  begin
    assert false report "I'm in the test_proc." severity note;
  end;

end for_pkg;

use work.for_pkg.all;

entity lower is
end lower;

architecture level of lower is
begin

 p1 : process
  begin
    test_proc;
    wait for 20 ns;
  end process;

end level;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture beh of top is

 component for_model
 end component;

 component lower
 end component;

begin
```

```
  linst1 : lower;

  linst2 : lower;

  finst  : for_model;

end beh;

configuration cfg_top of top is
 for beh
  for all : lower
   use entity work.lower(level);
  end for;
  for all : for_model
   use entity work.for_model(for_arch);
  end for;
 end for;
end cfg_top;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.for_pkg(body)
# Loading work.top(beh)
# Loading work.lower(level)
# Loading work.for_model(for_arch)
# Loading ./for_model.sl
# Foreign Init Function:
#   Region parameter is /top/finst; Primary name is for_model, Secondary
name is for_arch
# Load Done Callback Function:
#   Region /top; Primary name is top, Secondary name is beh
#   Region /top/linst1; Primary name is lower, Secondary name is level
#   Region /top/linst2; Primary name is lower, Secondary name is level
#   Region /top/finst; Primary name is for_model, Secondary name is
for_arch
#   Region /for_pkg; Primary name is for_pkg, Secondary name is <NULL>
VSIM 1> quit
```

mti_GetSignalMode()

Gets the mode (direction) of a signal.

Syntax

direction = mti_GetSignalMode(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal

Return Values

Name	Type	Description
direction	mtiDirectionT	The port mode of the specified signal

Description

mti_GetSignalMode() returns the direction (or port mode) of the specified VHDL signal. The direction is one of the following: MTI_INTERNAL, MTI_DIR_IN, MTI_DIR_OUT, or MTI_DIR_INOUT. MTI_INTERNAL indicates that the signal is not a port.

Examples

FLI code

```
#include <mti.h>

static char * convertDirection( mtiDirectionT direction )
{
  switch ( direction ) {
    case MTI_INTERNAL:  return "INTERNAL";
    case MTI_DIR_IN:    return "IN";
    case MTI_DIR_OUT:   return "OUT";
    case MTI_DIR_INOUT: return "INOUT";
    default:            return "UNKNOWN";
  }
}

void printSignals( mtiRegionIdT region, int indent )
{
  mtiSignalIdT sigid;

  for ( sigid = mti_FirstSignal( region ); sigid;
        sigid = mti_NextSignal() ) ) {
    if ( sigid ) {
      mti_PrintFormatted( "%*cSignal %s: Direction is %s\n",
                          indent, ' ', mti_GetSignalName( sigid ),
                          convertDirection( mti_GetSignalMode( sigid )));
    }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printSignals( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the    */
                              /* foreign attribute.                    */
```

```
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
  signal count : integer := 0;
begin
  b <= a after delay;

  p1 : process( a )
  begin
    count <= count + 1 after 0 ns;
  end process;
end b;

library ieee;
use ieee.std_logic_1164.all;

entity mid is
  port ( ptio : inout std_logic );
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );
```

```
  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

  p1 : process
    begin
      ptio <= 'U';
      wait for 1 ns;
      ptio <= 'Z';
      wait for 30 ns;
    end process;

end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is
  component mid is
    port ( ptio : inout std_logic );
  end component;

  signal sls : std_logic := '0';
begin
  inst1 : mid port map ( sls );
  sls <= std_logic'val( std_logic'pos(sls) + 1 ) after 5 ns;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl
# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Load Done phase:
#  Region /top
#    Signal sls: Direction is INTERNAL
#    Region /top/inst1
#      Signal ptio: Direction is INOUT
#      Signal s1: Direction is INTERNAL
#      Signal s2: Direction is INTERNAL
#      Signal s3: Direction is INTERNAL
#      Signal s4: Direction is INTERNAL
#      Region /top/inst1/flip
#        Signal a: Direction is IN
#        Signal b: Direction is OUT
#        Signal count: Direction is INTERNAL
#      Region /top/inst1/i1
#      Region /top/inst1/toggle
#        Signal a: Direction is IN
#        Signal b: Direction is OUT
#        Signal count: Direction is INTERNAL
VSIM 1> quit
```

mti_GetSignalName()

Gets the simple name of a scalar or top-level composite signal.

Syntax

signal_name = mti_GetSignalName(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal

Return Values

Name	Type	Description
signal_name	char *	The simple name of the signal

Description

mti_GetSignalName() returns the simple name of the specified VHDL or SystemC signal. If the signal is a composite subelement, then the name returned is the name of the top-level composite. You must not free the returned pointer.

To get the name of a composite subelement signal, use mti_GetSignalNameIndirect().

Examples

FLI code

```
#include <mti.h>

void printSignals( mtiRegionIdT region, int indent )
{
  mtiSignalIdT sigid;

  for ( sigid = mti_FirstSignal( region ); sigid;
        sigid = mti_NextSignal() ) {
    if ( sigid ) {
      mti_PrintFormatted( "%*cSignal %s\n",
                          indent, ' ', mti_GetSignalName( sigid ) );
    }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *      region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printSignals( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mtiSignalIdT * elem_list;
  mtiSignalIdT   sigid;

  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );

  mti_PrintMessage( "\nTesting names of composite subelements:\n" );
  sigid = mti_FindSignal( "/top/inst1/s3" );
  elem_list = mti_GetSignalSubelements( sigid, 0 );
  mti_PrintFormatted( "  Signal %s\n", mti_GetSignalName( elem_list[1] ) );
  mti_VsimFree( elem_list );
  sigid = mti_FindSignal( "/top/inst1/s4" );
  elem_list = mti_GetSignalSubelements( sigid, 0 );
  mti_PrintFormatted( "  Signal %s\n", mti_GetSignalName( elem_list[0] ) );
  mti_VsimFree( elem_list );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this   */
                             /* foreign architecture is instantiated. */
  char             *param,   /* The last part of the string in the    */
```

```
                                 /* foreign attribute.                 */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
  signal count : integer := 0;
begin
  b <= a after delay;

  p1 : process( a )
  begin
    count <= count + 1 after 0 ns;
  end process;
end b;

entity mid is

  type rectype is record
    a : integer;
    b : bit;
    c : bit_vector( 3 downto 0 );
  end record;

end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : rectype := ( 42, '1', "1100" );
  signal s4 : bit_vector( 7 downto 0 ) := "10001111";

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin
```

```
   i1 : for_model;

   s1 <= not s1 after 5 ns;

   toggle : inv port map ( s1, s2 );

end a;

entity top is
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Loading work.inv(b)
#
# Load Done phase:
#  Region /top
#    Region /top/inst1
#      Signal s1
#      Signal s2
#      Signal s3
#      Signal s4
#      Region /top/inst1/i1
#      Region /top/inst1/toggle
#        Signal a
#        Signal b
#        Signal count
#
# Testing names of composite subelements:
#   Signal s3
#   Signal s4
VSIM 1> quit
```

mti_GetSignalNameIndirect()

Gets the full simple name of a signal including array indices and record subelement names.

Syntax

signal_name = mti_GetSignalNameIndirect(signal_id, buffer, length)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal
buffer	char *	A buffer into which the signal name is to be placed; OPTIONAL - can be NULL
length	int	The length of the buffer parameter

Return Values

Name	Type	Description
signal_name	char *	The full simple name of the specified signal

Description

mti_GetSignalNameIndirect() returns the full simple name of the specified VHDL or SystemC signal including array indices and record fields. If the buffer parameter is NULL, then mti_GetSignalNameIndirect() allocates memory for the name and returns a pointer to it. The caller is responsible for freeing this memory with mti_VsimFree(). If the buffer parameter is not NULL, then mti_GetSignalNameIndirect() copies the name into the buffer parameter up to the length specified by the length parameter and also returns a pointer to the buffer parameter.

Examples

FLI code

```c
#include <mti.h>

static void printSignalInfo( mtiSignalIdT sigid, int indent )
{
 char        * signame;
 int           i;
 mtiSignalIdT * elem_list;
 mtiTypeIdT     sigtype;

 sigtype = mti_GetSignalType( sigid );
 signame = mti_GetSignalNameIndirect( sigid, 0, 0 );
 mti_PrintFormatted( "%*c%s\n", indent, ' ', signame );
 mti_VsimFree( signame );

 switch ( mti_GetTypeKind( sigtype ) ) {
  case MTI_TYPE_ARRAY:
   elem_list = mti_GetSignalSubelements( sigid, 0 );
   switch ( mti_GetTypeKind( mti_GetArrayElementType( sigtype )) ) {
    case MTI_TYPE_ARRAY:
    case MTI_TYPE_RECORD:
     for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
      printSignalInfo( elem_list[i], indent+2 );
     }
     break;
    default:
     for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
      signame = mti_GetSignalNameIndirect( elem_list[i], 0, 0 );
      mti_PrintFormatted( "%*c  %s\n", indent, ' ', signame );
      mti_VsimFree( signame );
     }
     break;
   }
   mti_VsimFree( elem_list );
   break;
  case MTI_TYPE_RECORD:
   elem_list = mti_GetSignalSubelements( sigid, 0 );
   for ( i = 0; i < mti_GetNumRecordElements( sigtype ); i++ ) {
    switch ( mti_GetTypeKind( mti_GetSignalType( elem_list[i] )) ) {
     case MTI_TYPE_ARRAY:
     case MTI_TYPE_RECORD:
      printSignalInfo( elem_list[i], indent+2 );
      break;
     default:
      signame = mti_GetSignalNameIndirect( elem_list[i], 0, 0 );
      mti_PrintFormatted( "%*c  %s\n", indent, ' ', signame );
      mti_VsimFree( signame );
      break;
    }
   }
   mti_VsimFree( elem_list );
   break;
  default:
   break;
  }
```

```
        }

        void loadDoneCB( void * param )
        {
         mti_PrintMessage( "\nComposite Signals:\n" );
         mti_PrintMessage( "  Signal /top/s1:" );
         printSignalInfo( mti_FindSignal( "/top/s1" ), 4 );
         mti_PrintMessage( "  Signal /top/s2:" );
         printSignalInfo( mti_FindSignal( "/top/s2" ), 4 );
         mti_PrintMessage( "  Signal /top/s3:" );
         printSignalInfo( mti_FindSignal( "/top/s3" ), 4 );
         mti_PrintMessage( "  Signal /top/s4:" );
         printSignalInfo( mti_FindSignal( "/top/s4" ), 4 );
         mti_PrintMessage( "  Signal /top/s5:" );
         printSignalInfo( mti_FindSignal( "/top/s5" ), 4 );
         mti_PrintMessage( "  Signal /top/s6:" );
         printSignalInfo( mti_FindSignal( "/top/s6" ), 4 );
        }

        void initForeign(
         mtiRegionIdT       region,   /* The ID of the region in which this    */
                                      /* foreign architecture is instantiated. */
         char              *param,    /* The last part of the string in the    */
                                      /* foreign attribute.                    */
         mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
         mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
        )
        {
         mti_AddLoadDoneCB( loadDoneCB, 0 );
        }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is

  type rectype is record
    a : integer;
    b : bit;
    c : bit_vector( 3 downto 0 );
  end record;

  type rectype2 is record
    f1 : bit;
    f2 : rectype;
  end record;

  type a1 is array ( 2 downto 0 ) of bit;
  type a2 is array ( 3 downto 2 ) of a1;
  type a3 is array ( 1 to 2, 0 to 4 ) of character;

end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : rectype := ( 42, '1', "1100" );
  signal s3 : bit_vector( 7 downto 0 ) := "10001111";
  signal s4 : rectype2 := ( '1', ( 16, '0', "1111" ) );
  signal s5 : a2 := ( "101", "011" );
  signal s6 : a3 := ( "Hello", "there" );

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Composite Signals:
#    Signal /top/s1:
#       s1
#    Signal /top/s2:
#       s2
#          s2.a
#          s2.b
#          s2.c
#            s2.c(3)
#            s2.c(2)
#            s2.c(1)
#            s2.c(0)
#    Signal /top/s3:
#       s3
#          s3(7)
#          s3(6)
#          s3(5)
#          s3(4)
#          s3(3)
#          s3(2)
#          s3(1)
#          s3(0)
#    Signal /top/s4:
#       s4
#          s4.f1
#          s4.f2
#            s4.f2.a
#            s4.f2.b
#            s4.f2.c
#               s4.f2.c(3)
#               s4.f2.c(2)
#               s4.f2.c(1)
#               s4.f2.c(0)
#    Signal /top/s5:
#       s5
#          s5(3)
#            s5(3)(2)
#            s5(3)(1)
#            s5(3)(0)
#          s5(2)
#            s5(2)(2)
#            s5(2)(1)
#            s5(2)(0)
#    Signal /top/s6:
#       s6
```

```
#        s6(1)
#          s6(1)(0)
#          s6(1)(1)
#          s6(1)(2)
#          s6(1)(3)
#          s6(1)(4)
#        s6(2)
#          s6(2)(0)
#          s6(2)(1)
#          s6(2)(2)
#          s6(2)(3)
#          s6(2)(4)
VSIM 1> quit
```

mti_GetSignalRegion()

Gets the region in which a signal is declared.

Syntax

region_id = mti_GetSignalRegion(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal

Return Values

Name	Type	Description
region_id	mtiRegionIdT	A handle to the region in which the specified signal is declared

Description

mti_GetSignalRegion() returns a handle to the region in which the specified VHDL or SystemC signal is declared.

If the signal is a port that has been collapsed, a handle to the region of the connected upper level signal is returned. Use the vsim option -nocollapse to disable the optimization of internal port map connections.

Examples

FLI code

```c
#include <mti.h>

void printSignals( mtiRegionIdT region, int indent )
{
  char          * region_name;
  mtiSignalIdT   sigid;

  for ( sigid = mti_FirstSignal( region ); sigid;
        sigid = mti_NextSignal() ) {
         region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ));
         mti_PrintFormatted( "%*cSignal %s is declared in region %s\n",
                             indent, ' ', mti_GetSignalName( sigid ),
                             region_name );
         mti_VsimFree( region_name );
        }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *         region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  mti_VsimFree( region_name );
  indent += 2;
  printSignals( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nHierarchy:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this     */
                                /* foreign architecture is instantiated.  */
  char               *param,    /* The last part of the string in the     */
                                /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
  signal count : integer := 0;
begin
  b <= a after delay;

  p1 : process( a )
  begin
    count <= count + 1 after 0 ns;
  end process;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;
```

```
      toggle : inv port map ( s1, s2 );

   end a;

   entity top is
   end top;

   architecture a of top is
     component mid is
     end component;
   begin
     inst1 : mid;
   end a;
```

Simulation output

```
   % vsim -c top
   Reading .../modeltech/tcl/vsim/pref.tcl

   # 5.4b

   # vsim -c top
   # Loading .../modeltech/sunos5/../std.standard
   # Loading work.top(a)
   # Loading work.mid(a)
   # Loading work.inv(b)
   # Loading work.for_model(a)
   # Loading ./for_model.sl
   #
   # Hierarchy:
   #  Region /top
   #    Region /top/inst1
   #      Signal s1 is declared in region /top/inst1
   #      Signal s2 is declared in region /top/inst1
   #      Signal s3 is declared in region /top/inst1
   #      Signal s4 is declared in region /top/inst1
   #      Region /top/inst1/flip
   #        Signal a is declared in region /top/inst1/flip
   #        Signal b is declared in region /top/inst1/flip
   #        Signal count is declared in region /top/inst1/flip
   #      Region /top/inst1/i1
   #      Region /top/inst1/toggle
   #        Signal a is declared in region /top/inst1/toggle
   #        Signal b is declared in region /top/inst1/toggle
   #        Signal count is declared in region /top/inst1/toggle
   VSIM 1> quit
```

mti_GetSignalSubelements()

Gets the subelements of a composite signal.

Syntax

elem_list = mti_GetSignalSubelements(signal_id, buffer);

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC composite signal
buffer	mtiSignalIdT *	A buffer into which the subelement signal IDs are to be placed;
		OPTIONAL - can be NULL

Return Values

Name	Type	Description
elem_list	mtiSignalIdT *	An array containing the signal IDs of the subelements of the specified signal

Description

mti_GetSignalSubelements() returns an array containing the signal IDs of the subelements of the specified VHDL or SystemC composite signal. If the buffer parameter is NULL, mti_GetSignalSubelements() allocates memory for the array and returns a pointer to it. The caller is responsible for freeing this memory with mti_VsimFree(). If the buffer parameter is not NULL, then mti_GetSignalSubelements() copies the subelement signal IDs into the buffer and also returns the buffer parameter. The length for the buffer parameter and the return value can be determined by calling mti_TickLength() on the type of the signal_id.

mti_GetSignalSubelements() returns NULL if the signal_id parameter is not a handle to a VHDL composite signal.

The internal representation of multi-dimensional arrays is the same as arrays of arrays. For example, array a(x,y,z) is accessed in the same manner as a(x)(y)(z). In order to get to the scalar subelements of an array of arrays, you must use mti_GetSignalSubelements() on each level of the array until reaching the scalar subelements.

Examples

FLI code

```
#include <mti.h>

static void printSignalInfo( mtiSignalIdT sigid, int indent )
{
  char         * signame;
  int            i;
  mtiSignalIdT * elem_list;
  mtiTypeIdT     sigtype;

  sigtype = mti_GetSignalType( sigid );
  signame = mti_GetSignalNameIndirect( sigid, 0, 0 );
  mti_PrintFormatted( "%*c%s\n", indent, ' ', signame );
  mti_VsimFree( signame );

  switch ( mti_GetTypeKind( sigtype ) ) {
   case MTI_TYPE_ARRAY:
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    switch ( mti_GetTypeKind( mti_GetArrayElementType( sigtype )) ) {
     case MTI_TYPE_ARRAY:
     case MTI_TYPE_RECORD:
      for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
        printSignalInfo( elem_list[i], indent+2 );
      }
      break;
     default:
      for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
        signame = mti_GetSignalNameIndirect( elem_list[i], 0, 0 );
        mti_PrintFormatted( "%*c  %s\n", indent, ' ', signame );
        mti_VsimFree( signame );
      }
      break;
    }
    mti_VsimFree( elem_list );
    break;
   case MTI_TYPE_RECORD:
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    for ( i = 0; i < mti_GetNumRecordElements( sigtype ); i++ ) {
      switch ( mti_GetTypeKind( mti_GetSignalType( elem_list[i] )) ) {
        case MTI_TYPE_ARRAY:
        case MTI_TYPE_RECORD:
          printSignalInfo( elem_list[i], indent+2 );
          break;
        default:
          signame = mti_GetSignalNameIndirect( elem_list[i], 0, 0 );
          mti_PrintFormatted( "%*c  %s\n", indent, ' ', signame );
          mti_VsimFree( signame );
          break;
      }
    }
    mti_VsimFree( elem_list );
    break;
   default:
    break;
  }
```

```
    }

    void loadDoneCB( void * param )
    {
      mti_PrintMessage( "\nComposite Signals:\n" );
      mti_PrintMessage( "  Signal /top/s1:" );
      printSignalInfo( mti_FindSignal( "/top/s1" ), 4 );
      mti_PrintMessage( "  Signal /top/s2:" );
      printSignalInfo( mti_FindSignal( "/top/s2" ), 4 );
      mti_PrintMessage( "  Signal /top/s3:" );
      printSignalInfo( mti_FindSignal( "/top/s3" ), 4 );
      mti_PrintMessage( "  Signal /top/s4:" );
      printSignalInfo( mti_FindSignal( "/top/s4" ), 4 );
      mti_PrintMessage( "  Signal /top/s5:" );
      printSignalInfo( mti_FindSignal( "/top/s5" ), 4 );
      mti_PrintMessage( "  Signal /top/s6:" );
      printSignalInfo( mti_FindSignal( "/top/s6" ), 4 );
    }

    void initForeign(
      mtiRegionIdT      region,   /* The ID of the region in which this    */
                                  /* foreign architecture is instantiated. */
      char             *param,    /* The last part of the string in the    */
                                  /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
    )
    {
      mti_AddLoadDoneCB( loadDoneCB, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is

  type rectype is record
    a : integer;
    b : bit;
    c : bit_vector( 3 downto 0 );
  end record;

  type rectype2 is record
    f1 : bit;
    f2 : rectype;
  end record;

  type a1 is array ( 2 downto 0 ) of bit;
  type a2 is array ( 3 downto 2 ) of a1;
  type a3 is array ( 1 to 2, 0 to 4 ) of character;

end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : rectype := ( 42, '1', "1100" );
  signal s3 : bit_vector( 7 downto 0 ) := "10001111";
  signal s4 : rectype2 := ( '1', ( 16, '0', "1111" ) );
  signal s5 : a2 := ( "101", "011" );
  signal s6 : a3 := ( "Hello", "there" );

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Composite Signals:
#   Signal /top/s1:
#     s1
#   Signal /top/s2:
#     s2
#       s2.a
#       s2.b
#       s2.c
#         s2.c(3)
#         s2.c(2)
#         s2.c(1)
#         s2.c(0)
#   Signal /top/s3:
#     s3
#       s3(7)
#       s3(6)
#       s3(5)
#       s3(4)
#       s3(3)
#       s3(2)
#       s3(1)
#       s3(0)
#   Signal /top/s4:
#     s4
#       s4.f1
#       s4.f2
#         s4.f2.a
#         s4.f2.b
#         s4.f2.c
#           s4.f2.c(3)
#           s4.f2.c(2)
#           s4.f2.c(1)
#           s4.f2.c(0)
#   Signal /top/s5:
#     s5
#       s5(3)
#         s5(3)(2)
#         s5(3)(1)
#         s5(3)(0)
#       s5(2)
#         s5(2)(2)
#         s5(2)(1)
#         s5(2)(0)
#   Signal /top/s6:
#     s6
```

```
#          s6(1)
#            s6(1)(0)
#            s6(1)(1)
#            s6(1)(2)
#            s6(1)(3)
#            s6(1)(4)
#          s6(2)
#            s6(2)(0)
#            s6(2)(1)
#            s6(2)(2)
#            s6(2)(3)
#            s6(2)(4)
VSIM 1> quit
```

mti_GetSignalType()

Gets the type of a signal.

Syntax

type_id = mti_GetSignalType(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the type ID of the specified signal

Description

mti_GetSignalType() returns a handle to the type ID of the specified VHDL or SystemC signal.

Examples

FLI code

```c
#include <mti.h>

static char * getTypeName( mtiTypeIdT type_id )
{
  switch ( mti_GetTypeKind( type_id ) ) {
    case MTI_TYPE_SCALAR:    return "SCALAR";
    case MTI_TYPE_ARRAY:     return "ARRAY";
    case MTI_TYPE_RECORD:    return "RECORD";
    case MTI_TYPE_ENUM:      return "ENUM";
    case MTI_TYPE_PHYSICAL:  return "PHYSICAL";
    case MTI_TYPE_REAL:      return "REAL";
    case MTI_TYPE_TIME:      return "TIME";
    default:                 return "UNKNOWN";
  }
}

static void printSignalInfo( mtiSignalIdT sigid, int indent )
{
  char          * signame;
  int             i;
  mtiSignalIdT * elem_list;
  mtiTypeIdT      sigtype;

  sigtype = mti_GetSignalType( sigid );
  signame = mti_GetSignalNameIndirect( sigid, 0, 0 );
  mti_PrintFormatted( "%*c%s is of type %s\n", indent, ' ', signame,
                      getTypeName( mti_GetSignalType( sigid )) );
  mti_VsimFree( signame );

  switch ( mti_GetTypeKind( sigtype ) ) {
   case MTI_TYPE_ARRAY:
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    switch ( mti_GetTypeKind( mti_GetArrayElementType( sigtype )) ) {
     case MTI_TYPE_ARRAY:
     case MTI_TYPE_RECORD:
      for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
        printSignalInfo( elem_list[i], indent+2 );
      }
      break;
     default:
      for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
        signame = mti_GetSignalNameIndirect( elem_list[i], 0, 0 );
        mti_PrintFormatted( "%*c%s is of type %s\n", indent, ' ', signame,
                            getTypeName( mti_GetSignalType( elem_list[i] )) );
        mti_VsimFree( signame );
      }
      break;
    }
    mti_VsimFree( elem_list );
    break;
   case MTI_TYPE_RECORD:
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    for ( i = 0; i < mti_GetNumRecordElements( sigtype ); i++ ) {
      switch ( mti_GetTypeKind( mti_GetSignalType( elem_list[i] )) ) {
```

```
                case MTI_TYPE_ARRAY:
                case MTI_TYPE_RECORD:
                  printSignalInfo( elem_list[i], indent+2 );
                  break;
                default:
                  signame = mti_GetSignalNameIndirect( elem_list[i], 0, 0 );
                  mti_PrintFormatted( "%*c%s is of type %s\n", indent, ' ', signame,
                                    getTypeName( mti_GetSignalType( elem_list[i] )) );
                  mti_VsimFree( signame );
                  break;
              }
            }
          mti_VsimFree( elem_list );
          break;
        default:
          break;
      }
    }

    void loadDoneCB( void * param )
    {
      mti_PrintMessage( "\nComposite Signals:\n" );
      mti_PrintMessage( "  Signal /top/s1:" );
      printSignalInfo( mti_FindSignal( "/top/s1" ), 4 );
      mti_PrintMessage( "  Signal /top/s2:" );
      printSignalInfo( mti_FindSignal( "/top/s2" ), 4 );
      mti_PrintMessage( "  Signal /top/s3:" );
      printSignalInfo( mti_FindSignal( "/top/s3" ), 4 );
      mti_PrintMessage( "  Signal /top/s4:" );
      printSignalInfo( mti_FindSignal( "/top/s4" ), 4 );
      mti_PrintMessage( "  Signal /top/s5:" );
      printSignalInfo( mti_FindSignal( "/top/s5" ), 4 );
      mti_PrintMessage( "  Signal /top/s6:" );
      printSignalInfo( mti_FindSignal( "/top/s6" ), 4 );
    }

    void initForeign(
      mtiRegionIdT      region,   /* The ID of the region in which this    */
                                  /* foreign architecture is instantiated. */
      char              *param,   /* The last part of the string in the    */
                                  /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
    )
    {
      mti_AddLoadDoneCB( loadDoneCB, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is

  type rectype is record
    a : integer;
    b : bit;
    c : bit_vector( 3 downto 0 );
  end record;

  type rectype2 is record
    f1 : bit;
    f2 : rectype;
  end record;

  type a1 is array ( 2 downto 0 ) of bit;
  type a2 is array ( 3 downto 2 ) of a1;
  type a3 is array ( 1 to 2, 0 to 4 ) of character;

end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : rectype := ( 42, '1', "1100" );
  signal s3 : bit_vector( 7 downto 0 ) := "10001111";
  signal s4 : rectype2 := ( '1', ( 16, '0', "1111" ) );
  signal s5 : a2 := ( "101", "011" );
  signal s6 : a3 := ( "Hello", "there" );

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Composite Signals:
#    Signal /top/s1:
#       s1 is of type ENUM
#    Signal /top/s2:
#       s2 is of type RECORD
#         s2.a is of type SCALAR
#         s2.b is of type ENUM
#         s2.c is of type ARRAY
#           s2.c(3) is of type ENUM
#           s2.c(2) is of type ENUM
#           s2.c(1) is of type ENUM
#           s2.c(0) is of type ENUM
#    Signal /top/s3:
#       s3 is of type ARRAY
#         s3(7) is of type ENUM
#         s3(6) is of type ENUM
#         s3(5) is of type ENUM
#         s3(4) is of type ENUM
#         s3(3) is of type ENUM
#         s3(2) is of type ENUM
#         s3(1) is of type ENUM
#         s3(0) is of type ENUM
#    Signal /top/s4:
#       s4 is of type RECORD
#         s4.f1 is of type ENUM
#         s4.f2 is of type RECORD
#           s4.f2.a is of type SCALAR
#           s4.f2.b is of type ENUM
#           s4.f2.c is of type ARRAY
#             s4.f2.c(3) is of type ENUM
#             s4.f2.c(2) is of type ENUM
#             s4.f2.c(1) is of type ENUM
#             s4.f2.c(0) is of type ENUM
#    Signal /top/s5:
#       s5 is of type ARRAY
#         s5(3) is of type ARRAY
#           s5(3)(2) is of type ENUM
#           s5(3)(1) is of type ENUM
#           s5(3)(0) is of type ENUM
#         s5(2) is of type ARRAY
#           s5(2)(2) is of type ENUM
#           s5(2)(1) is of type ENUM
#           s5(2)(0) is of type ENUM
#    Signal /top/s6:
#       s6 is of type ARRAY
```

```
#        s6(1) is of type ARRAY
#          s6(1)(0) is of type ENUM
#          s6(1)(1) is of type ENUM
#          s6(1)(2) is of type ENUM
#          s6(1)(3) is of type ENUM
#          s6(1)(4) is of type ENUM
#        s6(2) is of type ARRAY
#          s6(2)(0) is of type ENUM
#          s6(2)(1) is of type ENUM
#          s6(2)(2) is of type ENUM
#          s6(2)(3) is of type ENUM
#          s6(2)(4) is of type ENUM
VSIM 1> quit
```

mti_GetSignalValue()

Gets the value of a VHDL or SystemC scalar signal of type enumeration, integer, or physical (VHDL only).

Syntax

value = mti_GetSignalValue(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC scalar signal of type enumeration, integer, or physical (VHDL only)

Return Values

Name	Type	Description
value	mtiInt32T	The current value of the specified signal

Description

mti_GetSignalValue() returns the value of signals of type enumeration, integer, and physical (VHDL only). For composite, real, and time type signals, use mti_GetSignalValueIndirect().

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;     /* List of signals. */
  mtiProcessIdT  proc;         /* Test process id. */
} instanceInfoT;

static void printValue( mtiSignalIdT sigid, mtiTypeIdT sigtype, int indent )
{
  switch ( mti_GetTypeKind(sigtype) ) {
    case MTI_TYPE_ENUM:
      {
        char **  enum_values;
        mtiInt32T scalar_val;
        scalar_val  = mti_GetSignalValue( sigid );
        enum_values = mti_GetEnumValues( sigtype );
        mti_PrintFormatted( "  %s\n", enum_values[scalar_val] );
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetSignalValue( sigid );
        mti_PrintFormatted( "  %d\n", scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int          i;
        mtiInt32T    num_elems;
        mtiTypeIdT   elem_type;
        mtiTypeKindT elem_typekind;
        void       * array_val;

        array_val = mti_GetArraySignalValue( sigid, 0 );
        num_elems = mti_TickLength( sigtype );
        elem_type = mti_GetArrayElementType( sigtype );
        elem_typekind = mti_GetTypeKind( elem_type );
        switch ( elem_typekind ) {
          case MTI_TYPE_ENUM:
            {
              char ** enum_values;
              enum_values = mti_GetEnumValues( elem_type );
              if ( mti_TickLength( elem_type ) > 256 ) {
                mtiInt32T * val = array_val;
                  for ( i = 0; i < num_elems; i++ ) {
```

```
            mti_PrintFormatted( "  %s", enum_values[val[i]] );
          }
        } else {
          char * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %s", enum_values[val[i]] );
          }
        }
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  %d", val[i] );
        }
      }
      break;
    case MTI_TYPE_ARRAY:
      mti_PrintMessage( "  ARRAY" );
      break;
    case MTI_TYPE_RECORD:
      mti_PrintMessage( "  RECORD" );
      break;
    case MTI_TYPE_REAL:
      {
        double * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  %g", val[i] );
        }
      }
      break;
    case MTI_TYPE_TIME:
      {
        mtiTime64T * val = array_val;
        for ( i = 0; i < num_elems; i++ ) {
          mti_PrintFormatted( "  [%d,%d]",
                              MTI_TIME64_HI32(val[i]),
                              MTI_TIME64_LO32(val[i]) );
        }
      }
      break;
    default:
      break;
  }
  mti_PrintFormatted( "\n" );
  mti_VsimFree( array_val );
}
break;
case MTI_TYPE_RECORD:
  {
    int              i;
    mtiSignalIdT * elem_list;
    mtiInt32T       num_elems;
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    num_elems = mti_GetNumRecordElements( sigtype );
    mti_PrintFormatted( "\n" );
```

```
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "%*c", indent, ' ' );
            printValue( elem_list[i], mti_GetSignalType(elem_list[i]),
                        indent+2 );
          }
          mti_VsimFree( elem_list );
        }
        break;
      case MTI_TYPE_REAL:
        {
          double real_val;
          mti_GetSignalValueIndirect( sigid, &real_val );
          mti_PrintFormatted( "  %g\n", real_val );
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T time_val;
          mti_GetSignalValueIndirect( sigid, &time_val );
          mti_PrintFormatted( "  [%d,%d]\n",
                              MTI_TIME64_HI32(time_val),
                              MTI_TIME64_LO32(time_val) );
        }
        break;
      default:
        mti_PrintMessage( "\n" );
        break;
    }
}

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  signalInfoT   *siginfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
    mti_PrintFormatted( "  Signal %s:", siginfo->name );
    printValue( siginfo->sigid, siginfo->typeid, 4 );
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid  = sigid;
  siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid = mti_GetSignalType( sigid );
  siginfo->next   = 0;

  return( siginfo );
}
```

```
static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    siginfo = setupSignal( sigid );
    if ( inst_data->sig_info == 0 ) {
      inst_data->sig_info = siginfo;
    }
    else {
      curr_info->next = siginfo;
    }
      curr_info = siginfo;
  }

  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 6 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```vhdl
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig       <= not bitsig after 5 ns;
  intsig       <= intsig + 1 after 5 ns;
```

```
      physsig     <= physsig + 1 hour after 5 ns;
      realsig     <= realsig + 1.1 after 5 ns;
      timesig     <= timesig + 2 ns after 5 ns;
      stdlogicsig <= not stdlogicsig after 5 ns;

      stdlogicarr <= not stdlogicarr after 5 ns;

      rec.a       <= not rec.a after 5 ns;
      rec.b       <= rec.b + 1 after 5 ns;
      rec.c       <= not rec.c after 5 ns;

   end a;
```

Simulation output

```
      % vsim -c top
      Reading .../modeltech/tcl/vsim/pref.tcl

      # 5.4b

      # vsim -c top
      # Loading .../modeltech/sunos5/../std.standard
      # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
      # Loading work.top(a)
      # Loading work.for_model(a)
      # Loading ./for_model.sl
      VSIM 1> run 15
      # Time [0,6]:
      #   Signal bitsig:  '0'
      #   Signal intsig:  43
      #   Signal physsig:  4
      #   Signal realsig:  11.3
      #   Signal timesig:  [0,5]
      #   Signal stdlogicsig:  '0'
      #   Signal stdlogicarr:  '1'  '0'  '1'  '0'
      #   Signal rec:
      #       '1'
      #       1
      #       '0'  '1'  '1'  '0'
      # Time [0,11]:
      #   Signal bitsig:  '1'
      #   Signal intsig:  44
      #   Signal physsig:  5
      #   Signal realsig:  12.4
      #   Signal timesig:  [0,7]
      #   Signal stdlogicsig:  '1'
      #   Signal stdlogicarr:  '0'  '1'  '0'  '1'
      #   Signal rec:
      #       '0'
      #       2
      #       '1'  '0'  '0'  '1'
      VSIM 2> quit
```

mti_GetSignalValueIndirect()

Gets the value of a VHDL signal of any type except record.

Syntax

value = mti_GetSignalValueIndirect(signal_id, buffer)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL signal of any type except record
buffer	void *	A buffer into which the value is to be placed; OPTIONAL - can be NULL

Return Values

Name	Type	Description
value	void *	A pointer to the value of the specified signal

Description

mti_GetSignalValueIndirect() returns the value of a signal of any type except record. mti_GetSignalValueIndirect() must be used for scalar signals of type real and time.

If the buffer parameter is NULL, mti_GetSignalValueIndirect() allocates memory for the value and returns a pointer to it. The caller is responsible for freeing this memory with mti_VsimFree(). If the buffer parameter is not NULL, mti_GetSignalValueIndirect() copies the value into the buffer parameter and also returns the buffer parameter.

The returned value is interpreted as follows:

For a scalar signal or a subelement of type	The value should be cast to
Enum	(char *) if <= 256 values
	(mtiInt32T *) if > 256 values
Physical	(mtiInt32T *)
Real	(double *)
Scalar (Integer)	(mtiInt32T *)
Time	(mtiTime64T *)

In order to get the value of a record signal, use mti_GetSignalSubelements() to get handles to the signal subelements and then use mti_GetSignalValue(), mti_GetSignalValueIndirect(), or mti_GetArraySignalValue() on each of the subelements.

Examples

FLI code

```
#include <mti.h>

  typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;      /* List of signals. */
  mtiProcessIdT  proc;          /* Test process id. */
} instanceInfoT;

static void printValue( mtiSignalIdT sigid, mtiTypeIdT sigtype, int indent )
{
  switch ( mti_GetTypeKind(sigtype) ) {
    case MTI_TYPE_ENUM:
      {
        char **  enum_values;
        mtiInt32T scalar_val;
        scalar_val  = mti_GetSignalValue( sigid );
        enum_values = mti_GetEnumValues( sigtype );
        mti_PrintFormatted( "  %s\n", enum_values[scalar_val] );
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetSignalValue( sigid );
        mti_PrintFormatted( "  %d\n", scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int           i;
        mtiInt32T      num_elems;
        mtiTypeIdT     elem_type;
        mtiTypeKindT   elem_typekind;
        void         * array_val;

        array_val = mti_GetSignalValueIndirect( sigid, 0 );
        num_elems = mti_TickLength( sigtype );
        elem_type = mti_GetArrayElementType( sigtype );
        elem_typekind = mti_GetTypeKind( elem_type );
        switch ( elem_typekind ) {
          case MTI_TYPE_ENUM:
            {
              char ** enum_values;
              enum_values = mti_GetEnumValues( elem_type );
              if ( mti_TickLength( elem_type ) > 256 ) {
                mtiInt32T * val = array_val;
                  for ( i = 0; i < num_elems; i++ ) {
```

```
                mti_PrintFormatted( "  %s", enum_values[val[i]] );
              }
            } else {
              char * val = array_val;
              for ( i = 0; i < num_elems; i++ ) {
                mti_PrintFormatted( "  %s", enum_values[val[i]] );
              }
            }
          }
        }
        break;
      case MTI_TYPE_PHYSICAL:
      case MTI_TYPE_SCALAR:
        {
          mtiInt32T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %d", val[i] );
          }
        }
        break;
      case MTI_TYPE_ARRAY:
        mti_PrintMessage( "  ARRAY" );
        break;
      case MTI_TYPE_RECORD:
        mti_PrintMessage( "  RECORD" );
        break;
      case MTI_TYPE_REAL:
        {
          double * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %g", val[i] );
          }
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  [%d,%d]",
                               MTI_TIME64_HI32(val[i]),
                               MTI_TIME64_LO32(val[i]) );
          }
        }
        break;
      default:
        break;
    }
    mti_PrintFormatted( "\n" );
    mti_VsimFree( array_val );
  }
  break;
case MTI_TYPE_RECORD:
  {
    int            i;
    mtiSignalIdT * elem_list;
    mtiInt32T      num_elems;
    elem_list = mti_GetSignalSubelements( sigid, 0 );
    num_elems = mti_GetNumRecordElements( sigtype );
    mti_PrintFormatted( "\n" );
```

```
                for ( i = 0; i < num_elems; i++ ) {
                  mti_PrintFormatted( "%*c", indent, ' ' );
                  printValue( elem_list[i], mti_GetSignalType(elem_list[i]),
                             indent+2 );
                }
                mti_VsimFree( elem_list );
              }
            break;
          case MTI_TYPE_REAL:
            {
              double real_val;
              mti_GetSignalValueIndirect( sigid, &real_val );
              mti_PrintFormatted( "  %g\n", real_val );
            }
            break;
          case MTI_TYPE_TIME:
            {
              mtiTime64T time_val;
              mti_GetSignalValueIndirect( sigid, &time_val );
              mti_PrintFormatted( "  [%d,%d]\n",
                                 MTI_TIME64_HI32(time_val),
                                 MTI_TIME64_LO32(time_val) );
            }
            break;
          default:
            mti_PrintMessage( "\n" );
            break;
        }
      }

      static void checkValues( void *inst_info )
      {
        instanceInfoT *inst_data = (instanceInfoT *)inst_info;
        signalInfoT   *siginfo;

        mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

        for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
          mti_PrintFormatted( "  Signal %s:", siginfo->name );
          printValue( siginfo->sigid, siginfo->typeid, 4 );
        }

        mti_ScheduleWakeup( inst_data->proc, 5 );
      }

      static signalInfoT * setupSignal( mtiSignalIdT sigid )
      {
        signalInfoT * siginfo;

        siginfo        = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
        siginfo->sigid  = sigid;
        siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
        siginfo->typeid = mti_GetSignalType( sigid );
        siginfo->next   = 0;

        return( siginfo );
      }
```

```
static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data            = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    siginfo = setupSignal( sigid );
    if ( inst_data->sig_info == 0 ) {
      inst_data->sig_info = siginfo;
    }
    else {
      curr_info->next = siginfo;
    }
    curr_info = siginfo;
  }

  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 6 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;
  type intarray is array( 2 downto 0 ) of integer;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
  units
    hour;
    day   = 24 hour;
    week  = 7 day;
    month = 4 week;
    year  = 12 month;
  end units;

end top;

architecture a of top is

  signal bitsig      : bit        := '1';
  signal intsig      : integer    := 42;
  signal physsig     : bigtime    := 3 hour;
  signal realsig     : real       := 10.2;
  signal timesig     : time       := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal bitarr      : bitarray := "1100";
  signal intarr      : intarray := ( 5, 7, 9 );
  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
   bitsig      <= not bitsig after 5 ns;
   intsig      <= intsig + 1 after 5 ns;
   physsig     <= physsig + 1 hour after 5 ns;
   realsig     <= realsig + 1.1 after 5 ns;
   timesig     <= timesig + 2 ns after 5 ns;
   stdlogicsig <= not stdlogicsig after 5 ns;

   bitarr      <= not bitarr after 5 ns;
   stdlogicarr <= not stdlogicarr after 5 ns;

   intarr(2)   <= intarr(2) + 1 after 5 ns;
   intarr(1)   <= intarr(1) + 1 after 5 ns;
   intarr(0)   <= intarr(0) + 1 after 5 ns;

   rec.a       <= not rec.a after 5 ns;
   rec.b       <= rec.b + 1 after 5 ns;
   rec.c       <= not rec.c after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,6]:
#   Signal bitsig:  '0'
#   Signal intsig:  43
#   Signal physsig:  4
#   Signal realsig:  11.3
#   Signal timesig:  [0,5]
#   Signal stdlogicsig:  '0'
#   Signal bitarr:  '0'  '0'  '1'  '1'
#   Signal intarr:  6  8  10
#   Signal stdlogicarr:  '1'  '0'  '1'  '0'
#   Signal rec:
#       '1'
#       1
#       '0'  '1'  '1'  '0'
# Time [0,11]:
#   Signal bitsig:  '1'
#   Signal intsig:  44
#   Signal physsig:  5
#   Signal realsig:  12.4
#   Signal timesig:  [0,7]
#   Signal stdlogicsig:  '1'
#   Signal bitarr:  '1'  '1'  '0'  '0'
#   Signal intarr:  7  9  11
#   Signal stdlogicarr:  '0'  '1'  '0'  '1'
#   Signal rec:
#       '0'
#       2
#       '1'  '0'  '0'  '1'
VSIM 2> quit
```

mti_GetSimulationStatus()

Returns the current simulation status.

Syntax

int = mti_GetSimulationStatus()

Arguments

Return Values

Simulation status, such as:

- UCDB_TESTSTATUS_OK

- UCDB_TESTSTATUS_WARNING

- UCDB_TESTSTATUS_ERROR

mti_GetTopRegion()

Gets the first top-level region.

Syntax

region_id = mti_GetTopRegion()

Arguments

None

Return Values

Name	Type	Description
region_id	mtiRegionIdT	A handle to the first top-level region

Description

mti_GetTopRegion() returns the region ID of the first top-level region in the design hierarchy. You can use mti_NextRegion() to get additional top-level regions. Top-level regions are VHDL architectures and packages. Verilog modules, and SystemC sc_modules. If the region_id is a handle to a Verilog region, then you can use it with PLI functions to obtain information about and access objects in the Verilog region.

Examples

FLI code

```
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mtiRegionIdT regid;

  mti_PrintMessage( "\nDesign Hierarchy:\n" );
  for ( regid = mti_GetTopRegion(); regid; regid = mti_NextRegion(regid) ) {
    printHierarchy( regid, 1 );
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
top.vhd
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
          b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

package my_pkg is
   type my_type is array ( 7 downto 0 ) of integer;
end package my_pkg;

use work.my_pkg.all;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
            b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
```

```
      s3 <= not s3 after 5 ns;

      toggle : inv port map ( s1, s2 );

   end a;

   entity top is
   end top;

   architecture a of top is
     component mid is
     end component;
   begin
     inst1 : mid;
   end a;
   vertop.v
   module verbot;

      reg reg2;

      initial begin
        reg2 = 0;
      end

      always begin
        #5 reg2 = ~ reg2;
      end

   endmodule

   module vertop;

      reg reg1;

      initial begin
        reg1 = 0;
      end

      always begin
        #5 reg1 = ~ reg1;
      end

      verbot verinst1 ();

   endmodule
```

Simulation output

```
% vlog vertop.v
Model Technology ModelSim SE/EE vlog 5.4b Compiler 2000.06 Jun  9 2000
-- Compiling module verbot
-- Compiling module vertop

Top level modules:
  vertop
% vcom -93 top.vhd
Model Technology ModelSim SE/EE vcom 5.4b Compiler 2000.06 Jun  9 2000
-- Loading package standard
-- Compiling entity for_model
-- Compiling architecture a of for_model
-- Compiling entity inv
-- Compiling architecture b of inv
-- Compiling package my_pkg
-- Loading package my_pkg
-- Compiling entity mid
-- Compiling architecture a of mid
-- Loading entity for_model
-- Loading entity inv
-- Compiling entity top
-- Compiling architecture a of top
-- Loading entity mid
% vsim -c top vertop
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top vertop
# Loading .../modeltech/sunos5/../std.standard
# Loading work.my_pkg
# Loading work.top(a)
# Loading work.vertop
# Loading work.verbot
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Design Hierarchy:
#  Region /top
#    Region /top/inst1
#      Region /top/inst1/flip
#      Region /top/inst1/i1
#      Region /top/inst1/toggle
#  Region /vertop
#    Region /vertop/verinst1
#  Region /standard
#  Region /my_pkg
VSIM 1> quit
```

mti_GetTypeKind()

Gets the kind of a type.

Syntax

type_kind = mti_GetTypeKind(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
type_kind	mtiTypeKindT	The kind of the specified type

Description

mti_GetTypeKind() returns the kind of the specified VHDL or SystemC type. The returned value is one of the following:

type_kind	VHDL type
MTI_TYPE_SCALAR	Integer
MTI_TYPE_ARRAY	Array
MTI_TYPE_RECORD	Record
MTI_TYPE_ENUM	Enumeration
MTI_TYPE_PHYSICAL	Physical
MTI_TYPE_REAL	Real
MTI_TYPE_ACCESS	Access
MTI_TYPE_FILE	File
MTI_TYPE_TIME	Time

Examples

FLI code

```c
#include <mti.h>

static void printSignalInfo( mtiSignalIdT sigid, int indent )
{
  char        * signame;
  int           i;
  mtiSignalIdT * elem_list;
  mtiTypeIdT    sigtype;

  sigtype = mti_GetSignalType( sigid );
  signame = mti_GetSignalNameIndirect( sigid, 0, 0 );
  mti_PrintFormatted( "%*c%s ", indent, ' ', signame );
  mti_VsimFree( signame );


  switch ( mti_GetTypeKind( sigtype ) ) {
    case MTI_TYPE_SCALAR:
      mti_PrintFormatted( "is of type INTEGER\n" );
      break;
    case MTI_TYPE_ENUM:
      mti_PrintFormatted( "is of type ENUMERATION\n" );
      break;
    case MTI_TYPE_PHYSICAL:
      mti_PrintFormatted( "is of type PHYSICAL\n" );
      break;
    case MTI_TYPE_REAL:
      mti_PrintFormatted( "is of type REAL\n" );
      break;
    case MTI_TYPE_TIME:
      mti_PrintFormatted( "is of type TIME\n" );
      break;
    case MTI_TYPE_ARRAY:
      mti_PrintFormatted( "is of type ARRAY\n" );
      elem_list = mti_GetSignalSubelements( sigid, 0 );
      for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
        printSignalInfo( elem_list[i], indent+2 );
      }
      mti_VsimFree( elem_list );
      break;
    case MTI_TYPE_RECORD:
      mti_PrintFormatted( "is of type RECORD\n" );
      elem_list = mti_GetSignalSubelements( sigid, 0 );
      for ( i = 0; i < mti_GetNumRecordElements( sigtype ); i++ ) {
        printSignalInfo( elem_list[i], indent+2 );
      }
      mti_VsimFree( elem_list );
      break;
    default:
      mti_PrintFormatted( "is of type UNKNOWN\n" );
      break;
  }
}

void loadDoneCB( void * param )
```

```
    {
      mtiRegionIdT regid;
      mtiSignalIdT sigid;

      mti_PrintFormatted( "\nSignals:\n" );
      for ( regid = mti_GetTopRegion(); regid; regid = mti_NextRegion(regid) ) {
        for ( sigid = mti_FirstSignal( regid ); sigid;
              sigid = mti_NextSignal()) {
          printSignalInfo( sigid, 2 );
        }
      }
    }


    void initForeign(
      mtiRegionIdT        region,   /* The ID of the region in which this    */
                                    /* foreign architecture is instantiated. */
      char               *param,    /* The last part of the string in the    */
                                    /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
    )
    {
      mti_AddLoadDoneCB( loadDoneCB, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is

  type rectype is record
    a : integer;
    b : bit;
    c : bit_vector( 3 downto 0 );
  end record;

  type a1 is array ( 2 downto 0 ) of bit;
  type a2 is array ( 3 downto 2 ) of a1;

end top;

architecture a of top is

  signal s1 : bit := '0';
  signal s2 : rectype := ( 42, '1', "1100" );
  signal s3 : bit_vector( 7 downto 0 ) := "10001111";
  signal s5 : a2 := ( "101", "011" );
  signal s6 : integer := 42;
  signal s7 : real := 17.8;
  signal s8 : time := 11 ns;

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Signals:
#   s1 is of type ENUMERATION
#   s2 is of type RECORD
#     s2.a is of type INTEGER
#     s2.b is of type ENUMERATION
#     s2.c is of type ARRAY
#       s2.c(3) is of type ENUMERATION
#       s2.c(2) is of type ENUMERATION
#       s2.c(1) is of type ENUMERATION
#       s2.c(0) is of type ENUMERATION
#   s3 is of type ARRAY
#     s3(7) is of type ENUMERATION
#     s3(6) is of type ENUMERATION
#     s3(5) is of type ENUMERATION
#     s3(4) is of type ENUMERATION
#     s3(3) is of type ENUMERATION
#     s3(2) is of type ENUMERATION
#     s3(1) is of type ENUMERATION
#     s3(0) is of type ENUMERATION
#   s5 is of type ARRAY
#     s5(3) is of type ARRAY
#       s5(3)(2) is of type ENUMERATION
#       s5(3)(1) is of type ENUMERATION
#       s5(3)(0) is of type ENUMERATION
#     s5(2) is of type ARRAY
#       s5(2)(2) is of type ENUMERATION
#       s5(2)(1) is of type ENUMERATION
#       s5(2)(0) is of type ENUMERATION
#   s6 is of type INTEGER
#   s7 is of type REAL
#   s8 is of type TIME
VSIM 1> quit
```

mti_GetVarAddr()

Gets a pointer to a VHDL or SystemC variable's value space.

Syntax

value = mti_GetVarAddr(var_name)

Arguments

Name	Type	Description
var_name	char *	The name of a VHDL or SystemC variable

Return Values

Name	Type	Description
value	void *	A pointer to the value space of the specified variable

Description

mti_GetVarAddr() returns a pointer to the value space of a VHDL variable of any type except record.

You must specify the variable name according to the following rules:

- It can be either a full hierarchical name or a relative name. A relative name is relative to the region set by the environment command. The top-level region is the default.

- It must include the process label if the object is declared in a process.

- It must not include a slice specification.

The return is NULL if the variable is not found or if the variable is of a record type. You must not free the value pointer.

The value of the variable can be read and written at any time directly via the value pointer. The value pointer is interpreted as follows:

For a scalar variable or an array variable with a subelement of type	The value should be cast to
Enum	(char *) if <= 256 values
	(mtiInt32T *) if > 256 values
Physical	(mtiInt32T *)
Real	(double *)

For a scalar variable or an array variable with a subelement of type	The value should be cast to
Scalar (Integer)	(mtiInt32T *)
Time	(mtiTime64T *)

The number of subelements of an array variable can be determined by calling mti_TickLength() on the type of the array variable.

You can call mti_GetVarAddr() successfully only after elaboration is complete.

Examples

FLI code

```
#include <stdio.h>
#include <mti.h>

#define NAME_MAX 1024

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  void                 * var_addr;
  mtiVariableIdT         varid;
  mtiTypeIdT             typeid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;      /* List of variables. */
  mtiProcessIdT   proc;          /* Test process id. */
} instanceInfoT;

static void setValue( varInfoT * varinfo, int indent )
{
  switch ( mti_GetTypeKind( varinfo->typeid ) ) {
    case MTI_TYPE_ENUM:
      {
        char ** enum_values;
        enum_values = mti_GetEnumValues( varinfo->typeid );
        if ( mti_TickLength( varinfo->typeid ) <= 256 ) {
          char var_val = *(char *)(varinfo->var_addr);
          mti_PrintFormatted( "  %s\n", enum_values[(int)var_val] );
          var_val += 1;
          if (( var_val > mti_TickHigh( varinfo->typeid ) ) ||
              ( var_val < mti_TickLow( varinfo->typeid ) )) {
            var_val = mti_TickLeft( varinfo->typeid );
          }
          *(char *)(varinfo->var_addr) = var_val;
        } else {
          mtiInt32T var_val = *(mtiInt32T *)(varinfo->var_addr);
          mti_PrintFormatted( "  %s\n", enum_values[var_val] );
          var_val += 1;
          if (( var_val > mti_TickHigh( varinfo->typeid ) ) ||
              ( var_val < mti_TickLow( varinfo->typeid ) )) {
            var_val = mti_TickLeft( varinfo->typeid );
          }
          *(mtiInt32T *)(varinfo->var_addr) = var_val;
        }
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T var_val = *(mtiInt32T *)(varinfo->var_addr);
        mti_PrintFormatted( "  %d\n", var_val );
        var_val += 1;
        *(mtiInt32T *)(varinfo->var_addr) = var_val;
      }
```

```
            break;
        case MTI_TYPE_ARRAY:
          {
            int            i;
            mtiInt32T      num_elems;
            mtiTypeIdT     elem_type;
            mtiTypeKindT   elem_typekind;
            void         * array_val;

            array_val = varinfo->var_addr;
            num_elems = mti_TickLength( varinfo->typeid );
            elem_type = mti_GetArrayElementType( varinfo->typeid );
            elem_typekind = mti_GetTypeKind( elem_type );
            switch ( elem_typekind ) {
              case MTI_TYPE_ENUM:
                {
                  char ** enum_values;
                  enum_values = mti_GetEnumValues( elem_type );
                  if ( mti_TickLength( elem_type ) > 256 ) {
                    mtiInt32T * val = array_val;
                    for ( i = 0; i < num_elems; i++ ) {
                      mti_PrintFormatted( "  %s", enum_values[val[i]] );
                      val[i] += 1;
                      if (( val[i] > mti_TickHigh( elem_type )) ||
                          ( val[i] < mti_TickLow( elem_type ))) {
                        val[i] = mti_TickLeft( elem_type );
                      }
                    }
                  } else {
                    char * val = array_val;
                    for ( i = 0; i < num_elems; i++ ) {
                      mti_PrintFormatted( "  %s", enum_values[val[i]] );
                      val[i] += 1;
                      if (( val[i] > mti_TickHigh( elem_type )) ||
                          ( val[i] < mti_TickLow( elem_type ))) {
                        val[i] = mti_TickLeft( elem_type );
                      }
                    }
                  }
                }
                break;
              case MTI_TYPE_PHYSICAL:
              case MTI_TYPE_SCALAR:
                {
                  mtiInt32T * val = array_val;
                  for ( i = 0; i < num_elems; i++ ) {
                    mti_PrintFormatted( "  %d", val[i] );
                    val[i] += 1;
                  }
                }
                break;
              case MTI_TYPE_ARRAY:
                mti_PrintMessage( "  ARRAY" );
                break;
              case MTI_TYPE_RECORD:
                mti_PrintMessage( "  RECORD" );
                break;
              case MTI_TYPE_REAL:
```

```
            {
              double * val = array_val;
              for ( i = 0; i < num_elems; i++ ) {
                mti_PrintFormatted( "  %g", val[i] );
                val[i] += 1.1;
              }
            }
            break;
          case MTI_TYPE_TIME:
            {
              mtiTime64T * val = array_val;
              for ( i = 0; i < num_elems; i++ ) {
                mti_PrintFormatted( "  [%d,%d]",
                                    MTI_TIME64_HI32(val[i]),
                                    MTI_TIME64_LO32(val[i]) );
                MTI_TIME64_ASGN( val[i],
                                    MTI_TIME64_HI32(val[i]),
                                    MTI_TIME64_LO32(val[i]) + 1 );
              }
            }
            break;
          default:
            break;
          }
          mti_PrintFormatted( "\n" );
        }
        break;
    case MTI_TYPE_RECORD:
      mti_PrintFormatted( "  RECORD" );
      break;
    case MTI_TYPE_REAL:
      mti_PrintFormatted( "  %g\n", *(double *)(varinfo->var_addr) );
      *(double *)(varinfo->var_addr) += 1.1;
      break;
    case MTI_TYPE_TIME:
      {
        mtiTime64T time_val = *(mtiTime64T *)(varinfo->var_addr);
        mti_PrintFormatted( "  [%d,%d]\n",
                            MTI_TIME64_HI32(time_val),
                            MTI_TIME64_LO32(time_val) );
        MTI_TIME64_ASGN( *(mtiTime64T *)(varinfo->var_addr),
                            MTI_TIME64_HI32(time_val) + 1,
                            MTI_TIME64_LO32(time_val) + 1 );
      }
      break;
    default:
      mti_PrintMessage( "\n" );
      break;
  }
}

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  varInfoT      *varinfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );
```

```
      for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
        mti_PrintFormatted( "  Variable %s:", varinfo->name );
        setValue( varinfo, 4 );
      }

      mti_ScheduleWakeup( inst_data->proc, 5 );
   }

   static varInfoT * setupVariable(
     mtiVariableIdT varid,
     mtiRegionIdT   regid,
     mtiProcessIdT  procid
   )
   {
      char       var_name[NAME_MAX];
      char     * region_name;
      varInfoT * varinfo;

      varinfo            = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
      varinfo->varid     = varid;
      varinfo->name      = mti_GetVarName( varid );
      varinfo->typeid    = mti_GetVarType( varid );
      region_name        = mti_GetRegionFullName( regid );
      sprintf( var_name, "%s/%s/%s", region_name, mti_GetProcessName( procid ),
               varinfo->name );
      varinfo->var_addr = mti_GetVarAddr( var_name );
      mti_VsimFree( region_name );
      varinfo->next      = 0;

      return( varinfo );
   }

   static void initInstance( void * param )
   {
      instanceInfoT * inst_data;
      mtiProcessIdT   procid;
      mtiRegionIdT    regid;
      mtiVariableIdT  varid;
      varInfoT      * curr_info;
      varInfoT      * varinfo;

      inst_data            = mti_Malloc( sizeof(instanceInfoT) );
      inst_data->var_info = 0;

      regid = mti_GetTopRegion();
      for ( procid = mti_FirstProcess( regid );
            procid; procid = mti_NextProcess() ) {
        for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
          varinfo = setupVariable( varid, regid, procid );
          if ( inst_data->var_info == 0 ) {
            inst_data->var_info = varinfo;
          }
          else {
            curr_info->next = varinfo;
          }
          curr_info = varinfo;
        }
      }
```

```
      inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                           (void *)inst_data );
      mti_ScheduleWakeup( inst_data->proc, 5 );
   }

   void initForeign(
     mtiRegionIdT       region,   /* The ID of the region in which this   */
                                  /* foreign architecture is instantiated. */
     char              *param,    /* The last part of the string in the   */
                                  /* foreign attribute.                   */
     mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
     mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
   )
   {
     mti_AddLoadDoneCB( initInstance, 0 );
   }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic  := 'H';

    variable bitarr      : bitarray  := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray  := ( 10, 11, 12 );
    variable realarr     : realarray := ( 11.6, 101.22 );
    variable timearr     : timearray := ( 15 ns, 6 ns );

  begin

    wait for 5 ns;

  end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 16
# Time [0,5]:
#    Variable bitsig:  '1'
#    Variable intsig:  21
#    Variable realsig:  16.35
#    Variable timesig:  [0,5]
#    Variable stdlogicsig:  'H'
#    Variable bitarr:  '0'  '1'  '1'  '0'
#    Variable stdlogicarr:  '0'  '1'  'L'  'H'
#    Variable intarr:  10  11  12
#    Variable realarr:  11.6  101.22
#    Variable timearr:  [0,15]  [0,6]
# Time [0,10]:
#    Variable bitsig:  '0'
#    Variable intsig:  22
#    Variable realsig:  17.45
#    Variable timesig:  [1,6]
#    Variable stdlogicsig:  '-'
#    Variable bitarr:  '1'  '0'  '0'  '1'
#    Variable stdlogicarr:  '1'  'Z'  'H'  '-'
#    Variable intarr:  11  12  13
#    Variable realarr:  12.7  102.32
#    Variable timearr:  [0,16]  [0,7]
# Time [0,15]:
#    Variable bitsig:  '1'
#    Variable intsig:  23
#    Variable realsig:  18.55
#    Variable timesig:  [2,7]
#    Variable stdlogicsig:  'U'
#    Variable bitarr:  '0'  '1'  '1'  '0'
#    Variable stdlogicarr:  'Z'  'W'  '-'  'U'
#    Variable intarr:  12  13  14
#    Variable realarr:  13.8  103.42
#    Variable timearr:  [0,17]  [0,8]
VSIM 2> quit
```

mti_GetVarImage()

Gets the string image of the value of a VHDL constant, generic, or variable, or SystemC variable (by name).

Syntax

image = mti_GetVarImage(var_name)

Arguments

Name	Type	Description
var_name	char *	The name of a VHDL constant, generic, or variable, or SystemC variable.

Return Values

Name	Type	Description
image	char *	A string image of the value of the specified constant, generic, or variable

Description

mti_GetVarImage() returns a pointer to a buffer containing the string image of the value of the specified VHDL constant, generic, or variable. The image is the same as would be returned by the VHDL attribute 'IMAGE. The simulator returns NULL if the object is not found. The returned string is valid only until the next call to any FLI function. You must not free the returned pointer.

You must specify the name according to the following rules:

- It can be either a full hierarchical name or a relative name. A relative name is relative to the region set by the environment command. The top-level region is the default.

- It must include the process label if the object is declared in a process.

- It must not include a slice specification.

mti_GetVarImage() can be called successfully only after elaboration is complete.

Examples

FLI code

```
#include <stdio.h>
#include <mti.h>

#define NAME_MAX 1024

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiProcessIdT          procid;
  mtiRegionIdT           regid;
  mtiTypeIdT             typeid;
  mtiVariableIdT         varid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;     /* List of variables. */
  mtiProcessIdT   proc;         /* Test process id. */
} instanceInfoT;

static void checkValues( void *inst_info )
{
  char          * region_name;
  char            var_name[NAME_MAX];
  instanceInfoT * inst_data = (instanceInfoT *)inst_info;
  varInfoT      * varinfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
    region_name = mti_GetRegionFullName( varinfo->regid );
    sprintf( var_name, "%s/%s/%s", region_name,
            mti_GetProcessName( varinfo->procid ), varinfo->name );
    mti_PrintFormatted( "  Variable %s = %s\n",
                        var_name, mti_GetVarImage( var_name ));
    mti_VsimFree( region_name );
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable(
  mtiVariableIdT varid,
  mtiRegionIdT   regid,
  mtiProcessIdT  procid
)
{
  varInfoT * varinfo;

  varinfo        = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
  varinfo->varid  = varid;
  varinfo->name   = mti_GetVarName( varid );
  varinfo->typeid = mti_GetVarType( varid );
  varinfo->regid  = regid;
  varinfo->procid = procid;
```

```
      varinfo->next   = 0;

      return( varinfo );
   }

   static void initInstance( void * param )
   {
     instanceInfoT * inst_data;
     mtiProcessIdT   procid;
     mtiRegionIdT    regid;
     mtiVariableIdT  varid;
     varInfoT      * curr_info;
     varInfoT      * varinfo;

     inst_data          = mti_Malloc( sizeof(instanceInfoT) );
     inst_data->var_info = 0;

     regid = mti_GetTopRegion();
     for ( procid = mti_FirstProcess( regid );
           procid; procid = mti_NextProcess() ) {
       for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
         varinfo = setupVariable( varid, regid, procid );
         if ( inst_data->var_info == 0 ) {
           inst_data->var_info = varinfo;
         }
         else {
           curr_info->next = varinfo;
         }
         curr_info = varinfo;
       }
     }

     inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                          (void *)inst_data );
     mti_ScheduleWakeup( inst_data->proc, 4 );
   }

   void initForeign(
     mtiRegionIdT       region,   /* The ID of the region in which this   */
                                  /* foreign architecture is instantiated. */
     char              *param,    /* The last part of the string in the   */
                                  /* foreign attribute.                   */
     mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
     mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
   )
   {
     mti_AddLoadDoneCB( initInstance, 0 );
   }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : real;
    b : std_logic;
    c : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic  := 'H';

    variable bitarr      : bitarray   := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray   := ( 10, 11, 12 );
    variable realarr     : realarray  := ( 11.6, 101.22 );
    variable timearr     : timearray  := ( 15 ns, 6 ns );

    variable rec         : rectype    := ( 1.2, '0', "1001" );

  begin
    bitsig      := not bitsig;
    intsig      := intsig + 1;
```

```
        realsig    := realsig + 1.1;
        timesig    := timesig + 1 ns;
        stdlogicsig := not stdlogicsig;

        bitarr     := not bitarr;
        stdlogicarr := not stdlogicarr;

        intarr(1)  := intarr(1) + 1;
        intarr(2)  := intarr(2) + 1;
        intarr(3)  := intarr(3) + 1;

        realarr(1)  := realarr(1) + 1.1;
        realarr(2)  := realarr(2) + 1.1;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0) + 1 ns;

        rec.a      := rec.a + 1.1;
        rec.b      := not rec.b;
        rec.c      := not rec.c;

        wait for 5 ns;

    end process;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,4]:
#    Variable /top/p1/bitsig = '0'
#    Variable /top/p1/intsig = 22
#    Variable /top/p1/realsig = 1.745000e+01
#    Variable /top/p1/timesig = 6 ns
#    Variable /top/p1/stdlogicsig = '0'
#    Variable /top/p1/bitarr = "1001"
#    Variable /top/p1/stdlogicarr = "1010"
#    Variable /top/p1/intarr = (11, 12, 13)
#    Variable /top/p1/realarr = (1.270000e+01, 1.023200e+02)
#    Variable /top/p1/timearr = (16 ns, 7 ns)
#    Variable /top/p1/rec = (2.300000e+00, '1', "0110")
# Time [0,9]:
#    Variable /top/p1/bitsig = '1'
#    Variable /top/p1/intsig = 23
#    Variable /top/p1/realsig = 1.855000e+01
#    Variable /top/p1/timesig = 7 ns
#    Variable /top/p1/stdlogicsig = '1'
#    Variable /top/p1/bitarr = "0110"
#    Variable /top/p1/stdlogicarr = "0101"
#    Variable /top/p1/intarr = (12, 13, 14)
#    Variable /top/p1/realarr = (1.380000e+01, 1.034200e+02)
#    Variable /top/p1/timearr = (17 ns, 8 ns)
#    Variable /top/p1/rec = (3.400000e+00, '0', "1001")
VSIM 2> quit
```

mti_GetVarImageById()

Gets the string image of a VHDL or SystemC variable's value (by ID).

Syntax

image = mti_GetVarImageById(variable_id)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL or SystemC variable

Return Values

Name	Type	Description
image	char *	A string image of the specified variable's value

Description

mti_GetVarImageById() returns a pointer to a static buffer containing the string image of the specified VHDL variable's value. The image is the same as would be returned by the VHDL attribute 'IMAGE. The returned string is valid only until the next call to any FLI function. You must not free the returned pointer.

Examples

FLI code

```c
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiTypeIdT             typeid;
  mtiVariableIdT         varid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;     /* List of variables. */
  mtiProcessIdT   proc;         /* Test process id.*/
  } instanceInfoT;

static void checkValues( void *inst_info )
{
  instanceInfoT * inst_data = (instanceInfoT *)inst_info;
  varInfoT      * varinfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
    mti_PrintFormatted( "  Variable %s = %s\n",
                        varinfo->name,
                        mti_GetVarImageById( varinfo->varid ));
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable( mtiVariableIdT varid )
{
  varInfoT * varinfo;

  varinfo         = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
  varinfo->varid  = varid;
  varinfo->name   = mti_GetVarName( varid );
  varinfo->typeid = mti_GetVarType( varid );
  varinfo->next   = 0;

  return( varinfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiProcessIdT   procid;
  mtiRegionIdT    regid;
  mtiVariableIdT  varid;
  varInfoT      * curr_info;
  varInfoT      * varinfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->var_info = 0;
```

```
        regid = mti_GetTopRegion();
        for ( procid = mti_FirstProcess( regid );
              procid; procid = mti_NextProcess() ) {
          for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
            varinfo = setupVariable( varid );
            if ( inst_data->var_info == 0 ) {
              inst_data->var_info = varinfo;
            }
            else {
              curr_info->next = varinfo;
            }
            curr_info = varinfo;
          }
        }

        inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                          (void *)inst_data );
        mti_ScheduleWakeup( inst_data->proc, 4 );
      }

      void initForeign(
        mtiRegionIdT      region,   /* The ID of the region in which this     */
                                    /* foreign architecture is instantiated.  */
        char              *param,   /* The last part of the string in the     */
                                    /* foreign attribute.                     */
        mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
        mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
      )
      {
        mti_AddLoadDoneCB( initInstance, 0 );
      }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )    of integer;
  type realarray is array( 1 to 2 )    of real;
  type timearray is array( -1 to 0 )   of time;

  type rectype is record
    a : real;
    b : std_logic;
    c : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic  := 'H';

    variable bitarr      : bitarray   := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray   := ( 10, 11, 12 );
    variable realarr     : realarray  := ( 11.6, 101.22 );
    variable timearr     : timearray  := ( 15 ns, 6 ns );

    variable rec         : rectype    := ( 1.2, '0', "1001" );

  begin

    bitsig     := not bitsig;
```

```
        intsig      := intsig + 1;
        realsig     := realsig + 1.1;
        timesig     := timesig + 1 ns;
        stdlogicsig := not stdlogicsig;

        bitarr      := not bitarr;
        stdlogicarr := not stdlogicarr;

        intarr(1)   := intarr(1) + 1;
        intarr(2)   := intarr(2) + 1;
        intarr(3)   := intarr(3) + 1;

        realarr(1)  := realarr(1) + 1.1;
        realarr(2)  := realarr(2) + 1.1;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0) + 1 ns;

        rec.a       := rec.a + 1.1;
        rec.b       := not rec.b;
        rec.c       := not rec.c;

        wait for 5 ns;

    end process;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,4]:
#   Variable bitsig = '0'
#   Variable intsig = 22
#   Variable realsig = 1.745000e+01
#   Variable timesig = 6 ns
#   Variable stdlogicsig = '0'
#   Variable bitarr = "1001"
#   Variable stdlogicarr = "1010"
#   Variable intarr = (11, 12, 13)
#   Variable realarr = (1.270000e+01, 1.023200e+02)
#   Variable timearr = (16 ns, 7 ns)
#   Variable rec = (2.300000e+00, '1', "0110")
# Time [0,9]:
#   Variable bitsig = '1'
#   Variable intsig = 23
#   Variable realsig = 1.855000e+01
#   Variable timesig = 7 ns
#   Variable stdlogicsig = '1'
#   Variable bitarr = "0110"
#   Variable stdlogicarr = "0101"
#   Variable intarr = (12, 13, 14)
#   Variable realarr = (1.380000e+01, 1.034200e+02)
#   Variable timearr = (17 ns, 8 ns)
#   Variable rec = (3.400000e+00, '0', "1001")
VSIM 2> quit
```

mti_GetVarKind()

Gets the kind of VHDL variable.

Syntax

var_type = mti_GetVarKind(variable_id)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL variable

Return Values

Name	Type	Description
var_type	int	The type, but not fulltype, of the object. The value will be one of the acc values found in *acc_vhdl.h* and *acc_user.h* files

Description

mti_GetVarKind() returns the type, but not the fulltype, of the specified VHDL variable or NULL if no information can be found.

In general, you will use this function to distinguish between VHDL generics, variables and constants, where the return value will be accGeneric, accVariable, and accVHDLConstant, respectively. You should note that accVariable applies to both a regular VHDL variable and to a VHDL shared variable.

mti_GetVarName()

Gets the simple name of a VHDL or SystemC variable.

Syntax

var_name = mti_GetVarName(variable_id)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL or SystemC variable

Return Values

Name	Type	Description
var_name	char *	The simple name of the specified variable

Description

mti_GetVarName() returns the simple name of the specified VHDL variable or NULL if no information can be found. You must not free the returned pointer.

You cannot use mti_GetVarName() with variable IDs passed as foreign subprogram parameters.

Examples

FLI code

```c
#include <stdio.h>
#include <mti.h>

#define NAME_MAX 1024

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiProcessIdT          procid;
  mtiRegionIdT           regid;
  mtiTypeIdT             typeid;
  mtiVariableIdT         varid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;      /* List of variables. */
  mtiProcessIdT   proc;          /* Test process id. */
} instanceInfoT;

static void checkValues( void *inst_info )
{
 char           * region_name;
 char             var_name[NAME_MAX];
 instanceInfoT * inst_data = (instanceInfoT *)inst_info;
 varInfoT       * varinfo;

 mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

 for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
  region_name = mti_GetRegionFullName( varinfo->regid );
  sprintf( var_name, "%s/%s/%s", region_name,
          mti_GetProcessName( varinfo->procid ), varinfo->name );
  mti_PrintFormatted( "  Variable %s = %s\n",
                    var_name, mti_GetVarImageById( varinfo->varid ));
  mti_VsimFree( region_name );
 }

 mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable(
  mtiVariableIdT varid,
  mtiRegionIdT   regid,
  mtiProcessIdT  procid
)
{
  varInfoT * varinfo;

  varinfo        = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
  varinfo->varid  = varid;
  varinfo->name   = mti_GetVarName( varid );
  varinfo->typeid = mti_GetVarType( varid );
  varinfo->regid  = regid;
  varinfo->procid = procid;
```

```
      varinfo->next    = 0;

      return( varinfo );
   }

   static void initInstance( void * param )
   {
      instanceInfoT * inst_data;
      mtiProcessIdT   procid;
      mtiRegionIdT    regid;
      mtiVariableIdT  varid;
      varInfoT      * curr_info;
      varInfoT      * varinfo;

      inst_data          = mti_Malloc( sizeof(instanceInfoT) );
      inst_data->var_info = 0;

      regid = mti_GetTopRegion();
      for ( procid = mti_FirstProcess( regid );
            procid; procid = mti_NextProcess() ) {
        for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
          varinfo = setupVariable( varid, regid, procid );
          if ( inst_data->var_info == 0 ) {
            inst_data->var_info = varinfo;
          }
          else {
            curr_info->next = varinfo;
          }
          curr_info = varinfo;
        }
      }

      inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                          (void *)inst_data );
      mti_ScheduleWakeup( inst_data->proc, 4 );
   }

   void initForeign(
      mtiRegionIdT      region,   /* The ID of the region in which this    */
                                  /* foreign architecture is instantiated. */
      char             *param,    /* The last part of the string in the    */
                                  /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
   )
   {
      mti_AddLoadDoneCB( initInstance, 0 );
   }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : real;
    b : std_logic;
    c : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic  := 'H';

    variable bitarr      : bitarray   := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray   := ( 10, 11, 12 );
    variable realarr     : realarray  := ( 11.6, 101.22 );
    variable timearr     : timearray  := ( 15 ns, 6 ns );

    variable rec         : rectype    := ( 1.2, '0', "1001" );

  begin

    bitsig      := not bitsig;
```

```
        intsig      := intsig + 1;
        realsig     := realsig + 1.1;
        timesig     := timesig + 1 ns;
        stdlogicsig := not stdlogicsig;

        bitarr      := not bitarr;
        stdlogicarr := not stdlogicarr;

        intarr(1)   := intarr(1) + 1;
        intarr(2)   := intarr(2) + 1;
        intarr(3)   := intarr(3) + 1;

        realarr(1)  := realarr(1) + 1.1;
        realarr(2)  := realarr(2) + 1.1;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0) + 1 ns;

        rec.a       := rec.a + 1.1;
        rec.b       := not rec.b;
        rec.c       := not rec.c;

        wait for 5 ns;

    end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,4]:
#   Variable /top/p1/bitsig = '0'
#   Variable /top/p1/intsig = 22
#   Variable /top/p1/realsig = 1.745000e+01
#   Variable /top/p1/timesig = 6 ns
#   Variable /top/p1/stdlogicsig = '0'
#   Variable /top/p1/bitarr = "1001"
#   Variable /top/p1/stdlogicarr = "1010"
#   Variable /top/p1/intarr = (11, 12, 13)
#   Variable /top/p1/realarr = (1.270000e+01, 1.023200e+02)
#   Variable /top/p1/timearr = (16 ns, 7 ns)
#   Variable /top/p1/rec = (2.300000e+00, '1', "0110")
# Time [0,9]:
#   Variable /top/p1/bitsig = '1'
#   Variable /top/p1/intsig = 23
#   Variable /top/p1/realsig = 1.855000e+01
#   Variable /top/p1/timesig = 7 ns
#   Variable /top/p1/stdlogicsig = '1'
#   Variable /top/p1/bitarr = "0110"
#   Variable /top/p1/stdlogicarr = "0101"
#   Variable /top/p1/intarr = (12, 13, 14)
#   Variable /top/p1/realarr = (1.380000e+01, 1.034200e+02)
#   Variable /top/p1/timearr = (17 ns, 8 ns)
#   Variable /top/p1/rec = (3.400000e+00, '0', "1001")
VSIM 2> quit
```

mti_GetVarSubelements()

Gets the subelements of a composite VHDL variable. This does not support SystemC variables.

Syntax

elem_list = mti_GetVarSubelements(variable_id, buffer)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL variable
buffer	mtiVariableIdT *	A buffer into which the subelement variable IDs are to be placed; OPTIONAL - can be NULL

Return Values

Name	Type	Description
elem_list	mtiVariableIdT *	An array containing the variable IDs of the subelements of the specified variable

Description

mti_GetVarSubelements() returns an array containing the variable IDs of the subelements of the specified VHDL composite variable. If the buffer parameter is NULL, mti_GetVarSubelements() allocates memory for the array and returns a pointer to it. The caller is responsible for freeing this memory with mti_VsimFree(). If the buffer parameter is not NULL, then mti_GetVarSubelements() copies the subelement variable IDs into the buffer and also returns the buffer parameter. The length for the buffer parameter and the return value can be determined by calling mti_TickLength() on the type of the variable_id.

mti_GetVarSubelements() returns NULL if the variable_id parameter is not a handle to a VHDL composite variable.

The internal representation of multi-dimensional arrays is the same as arrays of arrays. For example, array a(x,y,z) is accessed in the same manner as a(x)(y)(z). In order to get to the scalar subelements of an array of arrays, you must use mti_GetVarSubelements() on each level of the array until reaching the scalar subelements.

Examples

FLI code

```c
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiVariableIdT         varid;
  mtiTypeIdT             typeid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;      /* List of variables. */
  mtiProcessIdT   proc;          /* Test process id. */
} instanceInfoT;

static void printValue(
  mtiVariableIdT varid,
  mtiTypeIdT     vartype,
  int            indent,
  int            print_newline
)
{
  switch ( mti_GetTypeKind( vartype ) ) {
    case MTI_TYPE_ENUM:
      {
        char ** enum_values;
        mtiInt32T scalar_val;
        enum_values = mti_GetEnumValues( vartype );
        scalar_val = mti_GetVarValue( varid );
        mti_PrintFormatted( "  %s", enum_values[scalar_val] );
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetVarValue( varid );
        mti_PrintFormatted( "  %d", scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int              i;
        mtiVariableIdT * elem_list;
        elem_list = mti_GetVarSubelements( varid, 0 );
        for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
          printValue( elem_list[i], mti_GetVarType(elem_list[i]),
                      indent, 0 );
        }
        mti_VsimFree( elem_list );
      }
      break;
    case MTI_TYPE_RECORD:
      {
        int              i;
```

```
                    mtiVariableIdT * elem_list;
                    mtiInt32T         num_elems;
                    elem_list = mti_GetVarSubelements( varid, 0 );
                    num_elems = mti_GetNumRecordElements( vartype );
                    mti_PrintFormatted( "\n" );
                    for ( i = 0; i < num_elems; i++ ) {
                      mti_PrintFormatted( "%*c", indent, ' ' );
                      printValue( elem_list[i], mti_GetVarType(elem_list[i]),
                               indent, 1 );
                    }
                    mti_VsimFree( elem_list );
                  }
                  break;
            case MTI_TYPE_REAL:
                  {
                    double real_val;
                    mti_GetVarValueIndirect( varid, &real_val );
                    mti_PrintFormatted( "  %g", real_val );
                  }
                  break;
            case MTI_TYPE_TIME:
                  {
                    mtiTime64T time_val;
                    mti_GetVarValueIndirect( varid, &time_val );
                    mti_PrintFormatted( "  [%d,%d]",
                                      MTI_TIME64_HI32(time_val),
                                      MTI_TIME64_LO32(time_val) );
                  }
                  break;
            default:
                  break;
        }
        if ( print_newline ) {
            mti_PrintFormatted( "\n" );
        }
}

static void checkValues( void *inst_info )
{
    instanceInfoT *inst_data = (instanceInfoT *)inst_info;
    varInfoT      *varinfo;

    mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

    for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
        mti_PrintFormatted( "  Variable %s:", varinfo->name );
        printValue( varinfo->varid, varinfo->typeid, 4, 1 );
    }

    mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable( mtiVariableIdT varid )
{
    varInfoT * varinfo;

    varinfo          = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
    varinfo->varid   = varid;
```

```
    varinfo->name    = mti_GetVarName( varid );
    varinfo->typeid  = mti_GetVarType( varid );
    varinfo->next    = 0;

  return( varinfo );
  }

  static void initInstance( void * param )
  {
    instanceInfoT * inst_data;
    mtiProcessIdT   procid;
    mtiVariableIdT  varid;
    varInfoT      * curr_info;
    varInfoT      * varinfo;

    inst_data          = mti_Malloc( sizeof(instanceInfoT) );
    inst_data->var_info = 0;

    for ( procid = mti_FirstProcess( mti_GetTopRegion() );
          procid; procid = mti_NextProcess() ) {
      for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
        varinfo = setupVariable( varid );
        if ( inst_data->var_info == 0 ) {
          inst_data->var_info = varinfo;
        }
        else {
          curr_info->next = varinfo;
        }
        curr_info = varinfo;
      }
    }

    inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                         (void *)inst_data );
    mti_ScheduleWakeup( inst_data->proc, 6 );
  }

  void initForeign(
    mtiRegionIdT      region,   /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
    char            *param,    /* The last part of the string in the    */
                                /* foreign attribute.                    */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
  )
  {
    mti_AddLoadDoneCB( initInstance, 0 );
  }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;


  type a1 is array ( 2 downto 0 ) of bitarray;
  type a2 is array ( 1 to 2, 3 to 4 ) of bitarray;


  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit       := '1';
    variable stdlogicsig : std_logic := 'H';

    variable bitarr      : bitarray  := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray  := ( 10, 11, 12 );
    variable realarr     : realarray := ( 11.6, 101.22 );
    variable timearr     : timearray := ( 15 ns, 6 ns );

    variable a1arr       : a1        := ( "1111", "0001", "0110" );
```

```
      variable a2arr        : a2        := ( ( "0001", "0010" ),
                                              ( "0100", "0101" ) );

      variable rec          : rectype   := ( '0', 1, 3.7, 'H', "1001" );

   begin

     bitsig      := not bitsig;
     stdlogicsig := not stdlogicsig;

     bitarr      := not bitarr;

     intarr(1)   := intarr(1) + 1;
     intarr(2)   := intarr(2) + 1;
     intarr(3)   := intarr(3) + 1;

     realarr(1)  := realarr(1) + 0.5;
     realarr(2)  := realarr(2) + 0.5;

     timearr(-1) := timearr(-1) + 1 ns;
     timearr(0)  := timearr(0)  + 1 ns;

     a1arr(2)    := not a1arr(2);
     a1arr(1)    := not a1arr(1);
     a1arr(0)    := not a1arr(0);
     a2arr(1,3)  := not a2arr(1,3);
     a2arr(1,4)  := not a2arr(1,4);
     a2arr(2,3)  := not a2arr(2,3);
     a2arr(2,4)  := not a2arr(2,4);

     stdlogicarr := not stdlogicarr;

     rec.a       := not rec.a;
     rec.b       := rec.b + 1;
     rec.c       := rec.c + 2.5;
     rec.d       := not rec.d;
     rec.e       := not rec.e;

     wait for 5 ns;

   end process;

   end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,6]:
#   Variable bitsig:  '1'
#   Variable stdlogicsig:  '1'
#   Variable bitarr:  '0'  '1'  '1'  '0'
#   Variable stdlogicarr:  '0'  '1'  '0'  '1'
#   Variable intarr:  12  13  14
#   Variable realarr:  12.6  102.22
#   Variable timearr:  [0,17]  [0,8]
#   Variable a1arr:  '1'  '1'  '1'  '1'  '0'  '0'  '0'  '1'  '0'  '1'  '1'
'0'
#   Variable a2arr:  '0'  '0'  '0'  '1'  '0'  '0'  '1'  '0'  '0'  '1'  '0'
'0'  '0'  '1'  '0'  '1'
#   Variable rec:
#       '0'
#        3
#        8.7
#       '1'
#       '1'  '0'  '0'  '1'
#
# Time [0,11]:
#   Variable bitsig:  '0'
#   Variable stdlogicsig:  '0'
#   Variable bitarr:  '1'  '0'  '0'  '1'
#   Variable stdlogicarr:  '1'  '0'  '1'  '0'
#   Variable intarr:  13  14  15
#   Variable realarr:  13.1  102.72
#   Variable timearr:  [0,18]  [0,9]
#   Variable a1arr:  '0'  '0'  '0'  '0'  '1'  '1'  '1'  '0'  '1'  '0'  '0'
'1'
#   Variable a2arr:  '1'  '1'  '1'  '0'  '1'  '1'  '0'  '1'  '1'  '0'  '1'
'1'  '1'  '0'  '1'  '0'
#   Variable rec:
#       '1'
#        4
#        11.2
#       '0'
#       '0'  '1'  '1'  '0'
#
VSIM 2> quit
```

mti_GetVarType()

Gets the type of a VHDL or SystemC variable.

Syntax

type_id = mti_GetVarType(variable_id)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL or SystemC variable

Return Values

Name	Type	Description
type_id	mtiTypeIdT	A handle to the type ID of the specified variable

Description

mti_GetVarType() returns a handle to the type of the specified VHDL variable.

Examples

FLI code

```
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  char * typestr;

  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:  typestr = "Scalar";  break;
    case MTI_TYPE_ARRAY:  typestr = "Array";  break;
    case MTI_TYPE_RECORD:  typestr = "Record";  break;
    case MTI_TYPE_ENUM:  typestr = "Enum";  break;
    case MTI_TYPE_PHYSICAL:  typestr = "Physical";  break;
    case MTI_TYPE_REAL:  typestr = "Real";  break;
    case MTI_TYPE_ACCESS:  typestr = "Access";  break;
    case MTI_TYPE_FILE:  typestr = "File";  break;
    case MTI_TYPE_TIME:  typestr = "Time";  break;
    default:  typestr = "UNKNOWN";  break;
  }

  return typestr;
}

static void printVarInfo( mtiVariableIdT varid )
{
  mti_PrintFormatted( "Variable %12s is of type %s\n",
                      mti_GetVarName( varid ),
                      getTypeStr( mti_GetVarType( varid )));
}

static void initInstance( void * param )
{
  mtiProcessIdT   procid;
  mtiRegionIdT    regid;
  mtiVariableIdT  varid;

  regid = mti_GetTopRegion();
  for ( procid = mti_FirstProcess( regid );
        procid; procid = mti_NextProcess() ) {
    for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
      printVarInfo( varid );
    }
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;

  type rectype is record
    a : real;
    b : std_logic;
    c : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin
  inst1 : for_model;

  p1 : process

    variable bitsig     : bit        := '1';
    variable intsig     : integer    := 21;
    variable realsig    : real       := 16.35;
    variable timesig    : time       := 5 ns;
    variable stdlogicsig : std_logic := 'H';

    variable bitarr     : bitarray   := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

    variable rec        : rectype    := ( 1.2, '0', "1001" );
  begin

    bitsig      := not bitsig;
    intsig      := intsig + 1;
    realsig     := realsig + 1.1;
    timesig     := timesig + 1 ns;
    stdlogicsig := not stdlogicsig;

    bitarr      := not bitarr;
    stdlogicarr := not stdlogicarr;
```

```
        rec.a       := rec.a + 1.1;
        rec.b       := not rec.b;
        rec.c       := not rec.c;

        wait for 5 ns;

    end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Variable      bitsig is of type Enum
# Variable      intsig is of type Scalar
# Variable     realsig is of type Real
# Variable     timesig is of type Time
# Variable  stdlogicsig is of type Enum
# Variable      bitarr is of type Array
# Variable  stdlogicarr is of type Array
# Variable         rec is of type Record
VSIM 1> quit
```

mti_GetVarValue()

Gets the value of a scalar VHDL or SystemC variable of type enumeration, integer, or physical.

Syntax

value = mti_GetVarValue(variable_id)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL scalar variable of type enumeration, integer, or physical, or SystemC variable.

Return Values

Name	Type	Description
value	mtiInt32T	The current value of the specified variable

Description

mti_GetVarValue() returns the value of variables of type enumeration, integer, and physical. For composite, real, and time type variables, use mti_GetVarValueIndirect().

Examples

FLI code

```c
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiVariableIdT         varid;
  mtiTypeIdT             typeid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;      /* List of variables. */
  mtiProcessIdT   proc;          /* Test process id. */
} instanceInfoT;

static void printValue( mtiVariableIdT varid, mtiTypeIdT vartype, int indent )
{
  switch ( mti_GetTypeKind( vartype ) ) {
    case MTI_TYPE_ENUM:
      {
        char ** enum_values;
        mtiInt32T scalar_val;
        enum_values = mti_GetEnumValues( vartype );
        scalar_val = mti_GetVarValue( varid );
        mti_PrintFormatted( "  %s\n", enum_values[scalar_val] );
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetVarValue( varid );
        mti_PrintFormatted( "  %d\n", scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int            i;
        mtiInt32T      num_elems;
        mtiTypeIdT     elem_type;
        mtiTypeKindT   elem_typekind;
        void         * array_val;

        array_val = mti_GetArrayVarValue( varid, 0 );
        num_elems = mti_TickLength( vartype );
        elem_type = mti_GetArrayElementType( vartype );
        elem_typekind = mti_GetTypeKind( elem_type );
        switch ( elem_typekind ) {
          case MTI_TYPE_ENUM:
            {
              char ** enum_values;
              enum_values = mti_GetEnumValues( elem_type );
              if ( mti_TickLength( elem_type ) > 256 ) {
                mtiInt32T * val = array_val;
                  for ( i = 0; i < num_elems; i++ ) {
```

```
              mti_PrintFormatted( "  %s", enum_values[val[i]] );
            }
          } else {
            char * val = array_val;
            for ( i = 0; i < num_elems; i++ ) {
              mti_PrintFormatted( "  %s", enum_values[val[i]] );
            }
          }
        }
        break;
      case MTI_TYPE_PHYSICAL:
      case MTI_TYPE_SCALAR:
        {
          mtiInt32T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %d", val[i] );
          }
        }
        break;
      case MTI_TYPE_ARRAY:
        mti_PrintMessage( "  ARRAY" );
        break;
      case MTI_TYPE_RECORD:
        mti_PrintMessage( "  RECORD" );
        break;
      case MTI_TYPE_REAL:
        {
          double * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  %g", val[i] );
          }
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T * val = array_val;
          for ( i = 0; i < num_elems; i++ ) {
            mti_PrintFormatted( "  [%d,%d]",
                                MTI_TIME64_HI32(val[i]),
                                MTI_TIME64_LO32(val[i]) );
          }
        }
        break;
      default:
        break;
    }
    mti_PrintFormatted( "\n" );
  }
  break;
case MTI_TYPE_RECORD:
  {
    int              i;
    mtiVariableIdT * elem_list;
    mtiInt32T        num_elems;
    elem_list = mti_GetVarSubelements( varid, 0 );
    num_elems = mti_GetNumRecordElements( vartype );
    mti_PrintFormatted( "\n" );
    for ( i = 0; i < num_elems; i++ ) {
```

```
                mti_PrintFormatted( "%*c", indent, ' ' );
                printValue( elem_list[i], mti_GetVarType(elem_list[i]),
                            indent+2 );
              }
              mti_VsimFree( elem_list );
            }
            break;
        case MTI_TYPE_REAL:
          {
            double real_val;
            mti_GetVarValueIndirect( varid, &real_val );
            mti_PrintFormatted( "  %g\n", real_val );
          }
          break;
        case MTI_TYPE_TIME:
          {
            mtiTime64T time_val;
            mti_GetVarValueIndirect( varid, &time_val );
            mti_PrintFormatted( "  [%d,%d]\n",
                                MTI_TIME64_HI32(time_val),
                                MTI_TIME64_LO32(time_val) );
          }
          break;
        default:
          mti_PrintMessage( "\n" );
          break;
      }
    }

    static void checkValues( void *inst_info )
    {
      instanceInfoT *inst_data = (instanceInfoT *)inst_info;
      varInfoT      *varinfo;

      mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

      for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
        mti_PrintFormatted( "  Variable %s:", varinfo->name );
        printValue( varinfo->varid, varinfo->typeid, 4 );
      }

      mti_ScheduleWakeup( inst_data->proc, 5 );
    }

    static varInfoT * setupVariable( mtiVariableIdT varid )
    {
      varInfoT * varinfo;

      varinfo         = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
      varinfo->varid  = varid;
      varinfo->name   = mti_GetVarName( varid );
      varinfo->typeid = mti_GetVarType( varid );
      varinfo->next   = 0;

      return( varinfo );
    }

    static void initInstance( void * param )
```

```
{
  instanceInfoT * inst_data;
  mtiProcessIdT   procid;
  mtiVariableIdT  varid;
  varInfoT      * curr_info;
  varInfoT      * varinfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->var_info = 0;

  for ( procid = mti_FirstProcess( mti_GetTopRegion() );
        procid; procid = mti_NextProcess() ) {
    for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
      varinfo = setupVariable( varid );
      if ( inst_data->var_info == 0 ) {
        inst_data->var_info = varinfo;
      }
      else {
        curr_info->next = varinfo;
      }
      curr_info = varinfo;
    }
  }

  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 6 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic  := 'H';

    variable bitarr      : bitarray   := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray   := ( 10, 11, 12 );
    variable realarr     : realarray  := ( 11.6, 101.22 );
    variable timearr     : timearray  := ( 15 ns, 6 ns );

    variable rec         : rectype    := ( '0', 1, 3.7, 'H', "1001" );

  begin
```

```
        bitsig      := not bitsig;
        intsig      := intsig + 1;
        realsig     := realsig + 1.5;
        timesig     := timesig + 1 ns;
        stdlogicsig := not stdlogicsig;

        bitarr      := not bitarr;

        intarr(1)   := intarr(1) + 1;
        intarr(2)   := intarr(2) + 1;
        intarr(3)   := intarr(3) + 1;

        realarr(1)  := realarr(1) + 0.5;
        realarr(2)  := realarr(2) + 0.5;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0)  + 1 ns;

        stdlogicarr := not stdlogicarr;

        rec.a       := not rec.a;
        rec.b       := rec.b + 1;
        rec.c       := rec.c + 2.5;
        rec.d       := not rec.d;
        rec.e       := not rec.e;

        wait for 5 ns;

    end process;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,6]:
#   Variable bitsig:  '1'
#   Variable intsig:  23
#   Variable realsig:  19.35
#   Variable timesig:  [0,7]
#   Variable stdlogicsig:  '1'
#   Variable bitarr:  '0'  '1'  '1'  '0'
#   Variable stdlogicarr:  '0'  '1'  '0'  '1'
#   Variable intarr:  12  13  14
#   Variable realarr:  12.6  102.22
#   Variable timearr:  [0,17]  [0,8]
#   Variable rec:
#       '0'
#       3
#       8.7
#       '1'
#       '1'  '0'  '0'  '1'
# Time [0,11]:
#   Variable bitsig:  '0'
#   Variable intsig:  24
#   Variable realsig:  20.85
#   Variable timesig:  [0,8]
#   Variable stdlogicsig:  '0'
#   Variable bitarr:  '1'  '0'  '0'  '1'
#   Variable stdlogicarr:  '1'  '0'  '1'  '0'
#   Variable intarr:  13  14  15
#   Variable realarr:  13.1  102.72
#   Variable timearr:  [0,18]  [0,9]
#   Variable rec:
#       '1'
#       4
#       11.2
#       '0'
#       '0'  '1'  '1'  '0'
VSIM 2> quit
```

mti_GetVarValueIndirect()

Gets the value of a VHDL variable of any type except record or SystemC variable.

Syntax

value = mti_GetVarValueIndirect(variable_id, buffer)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL variable of any type except record or SystemC variable.
buffer	void *	A buffer into which the value is to be placed; OPTIONAL - can be NULL

Return Values

Name	Type	Description
value	void *	A pointer to the value of the specified variable

Description

mti_GetVarValueIndirect() returns the value of a variable of any type except record. mti_GetVarValueIndirect() must be used for scalar variables of type real and time.

If the buffer parameter is NULL, mti_GetVarValueIndirect() returns a pointer to the value, which must be treated as read-only data and must not be freed.

If the buffer parameter is not NULL, mti_GetVarValueIndirect() copies the value in the buffer parameter and also returns the buffer parameter.

The returned value is interpreted as follows:

For a scalar variable or a subelement of type	The value should be cast to
Enum	(char *) if <= 256 values
	(mtiInt32T *) if > 256 values
Physical	(mtiInt32T *)
Real	(double *)
Scalar (Integer)	(mtiInt32T *)
Time	(mtiTime64T *)

In order to get the value of a record variable, use mti_GetVarSubelements() to get handles to the variable subelements and then use mti_GetVarValue(), mti_GetVarValueIndirect(), or mti_GetArrayVarValue() on each of the subelements.

Examples

FLI code

```c
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiVariableIdT         varid;
  mtiTypeIdT             typeid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;      /* List of variables. */
  mtiProcessIdT   proc;          /* Test process id. */
} instanceInfoT;

static void printValue( mtiVariableIdT varid, mtiTypeIdT vartype, int indent )
{
  switch ( mti_GetTypeKind( vartype ) ) {
    case MTI_TYPE_ENUM:
      {
        char ** enum_values;
        enum_values = mti_GetEnumValues( vartype );
        if ( mti_TickLength( vartype ) > 256 ) {
          mtiInt32T scalar_val;
          (void) mti_GetVarValueIndirect( varid, &scalar_val );
          mti_PrintFormatted( "  %s\n", enum_values[scalar_val] );
        } else {
          char scalar_val;
          (void) mti_GetVarValueIndirect( varid, &scalar_val );
          mti_PrintFormatted( "  %s\n", enum_values[(int)scalar_val] );
        }
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetVarValue( varid );
        mti_PrintFormatted( "  %d\n", scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int            i;
        mtiInt32T      num_elems;
        mtiTypeIdT     elem_type;
        mtiTypeKindT   elem_typekind;
        void         * array_val;

        array_val = mti_GetArrayVarValue( varid, 0 );
        num_elems = mti_TickLength( vartype );
        elem_type = mti_GetArrayElementType( vartype );
        elem_typekind = mti_GetTypeKind( elem_type );
        switch ( elem_typekind ) {
          case MTI_TYPE_ENUM:
```

```
          {
            char ** enum_values;
            enum_values = mti_GetEnumValues( elem_type );
            if ( mti_TickLength( elem_type ) > 256 ) {
              mtiInt32T * val = array_val;
              for ( i = 0; i < num_elems; i++ ) {
                mti_PrintFormatted( "  %s", enum_values[val[i]] );
              }
            } else {
              char * val = array_val;
              for ( i = 0; i < num_elems; i++ ) {
                mti_PrintFormatted( "  %s", enum_values[val[i]] );
              }
            }
          }
          break;
        case MTI_TYPE_PHYSICAL:
        case MTI_TYPE_SCALAR:
          {
            mtiInt32T * val = array_val;
            for ( i = 0; i < num_elems; i++ ) {
              mti_PrintFormatted( "  %d", val[i] );
            }
          }
          break;
        case MTI_TYPE_ARRAY:
          mti_PrintMessage( "  ARRAY" );
          break;
        case MTI_TYPE_RECORD:
          mti_PrintMessage( "  RECORD" );
          break;
        case MTI_TYPE_REAL:
          {
            double * val = array_val;
            for ( i = 0; i < num_elems; i++ ) {
              mti_PrintFormatted( "  %g", val[i] );
            }
          }
          break;
        case MTI_TYPE_TIME:
          {
            mtiTime64T * val = array_val;
            for ( i = 0; i < num_elems; i++ ) {
              mti_PrintFormatted( "  [%d,%d]",
                                  MTI_TIME64_HI32(val[i]),
                                  MTI_TIME64_LO32(val[i]) );
            }
          }
          break;
        default:
          break;
      }
      mti_PrintFormatted( "\n" );
    }
    break;
  case MTI_TYPE_RECORD:
    {
      int             i;
```

```
            mtiVariableIdT * elem_list;
            mtiInt32T        num_elems;
            elem_list = mti_GetVarSubelements( varid, 0 );
            num_elems = mti_GetNumRecordElements( vartype );
            mti_PrintFormatted( "\n" );
            for ( i = 0; i < num_elems; i++ ) {
              mti_PrintFormatted( "%*c", indent, ' ' );
              printValue( elem_list[i], mti_GetVarType(elem_list[i]),
                        indent+2 );
            }
            mti_VsimFree( elem_list );
          }
        break;
      case MTI_TYPE_REAL:
        {
          double real_val;
          mti_GetVarValueIndirect( varid, &real_val );
          mti_PrintFormatted( "  %g\n", real_val );
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T time_val;
          mti_GetVarValueIndirect( varid, &time_val );
          mti_PrintFormatted( "  [%d,%d]\n",
                              MTI_TIME64_HI32(time_val),
                              MTI_TIME64_LO32(time_val) );
        }
        break;
      default:
        mti_PrintMessage( "\n" );
        break;
    }
  }

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  varInfoT      *varinfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
    mti_PrintFormatted( "  Variable %s:", varinfo->name );
    printValue( varinfo->varid, varinfo->typeid, 4 );
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable( mtiVariableIdT varid )
{
  varInfoT * varinfo;

  varinfo          = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
  varinfo->varid   = varid;
  varinfo->name    = mti_GetVarName( varid );
  varinfo->typeid  = mti_GetVarType( varid );
```

```
      varinfo->next    = 0;

      return( varinfo );
}

static void initInstance( void * param )
{
   instanceInfoT * inst_data;
   mtiProcessIdT   procid;
   mtiVariableIdT  varid;
   varInfoT       * curr_info;
   varInfoT       * varinfo;

   inst_data          = mti_Malloc( sizeof(instanceInfoT) );
   inst_data->var_info = 0;

   for ( procid = mti_FirstProcess( mti_GetTopRegion() );
         procid; procid = mti_NextProcess() ) {
     for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
       varinfo = setupVariable( varid );
       if ( inst_data->var_info == 0 ) {
         inst_data->var_info = varinfo;
       }
       else {
         curr_info->next = varinfo;
       }
       curr_info = varinfo;
     }
   }

   inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                        (void *)inst_data );
   mti_ScheduleWakeup( inst_data->proc, 6 );
}

void initForeign(
   mtiRegionIdT      region,   /* The ID of the region in which this   */
                               /* foreign architecture is instantiated. */
   char             *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
   mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
   mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
   mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic := 'H';

    variable bitarr      : bitarray   := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray   := ( 10, 11, 12 );
    variable realarr     : realarray := ( 11.6, 101.22 );
    variable timearr     : timearray := ( 15 ns, 6 ns );

    variable rec         : rectype    := ( '0', 1, 3.7, 'H', "1001" );

  begin
```

```
        bitsig      := not bitsig;
        intsig      := intsig + 1;
        realsig     := realsig + 1.5;
        timesig     := timesig + 1 ns;
        stdlogicsig := not stdlogicsig;

        bitarr      := not bitarr;

        intarr(1)   := intarr(1) + 1;
        intarr(2)   := intarr(2) + 1;
        intarr(3)   := intarr(3) + 1;

        realarr(1)  := realarr(1) + 0.5;
        realarr(2)  := realarr(2) + 0.5;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0)  + 1 ns;

        stdlogicarr := not stdlogicarr;

        rec.a       := not rec.a;
        rec.b       := rec.b + 1;
        rec.c       := rec.c + 2.5;
        rec.d       := not rec.d;
        rec.e       := not rec.e;

        wait for 5 ns;

    end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,6]:
#    Variable bitsig:  '1'
#    Variable intsig:  23
#    Variable realsig:  19.35
#    Variable timesig:  [0,7]
#    Variable stdlogicsig:  '1'
#    Variable bitarr:  '0'  '1'  '1'  '0'
#    Variable stdlogicarr:  '0'  '1'  '0'  '1'
#    Variable intarr:  12  13  14
#    Variable realarr:  12.6  102.22
#    Variable timearr:  [0,17]  [0,8]
#    Variable rec:
#        '0'
#        3
#        8.7
#        '1'
#        '1'  '0'  '0'  '1'
# Time [0,11]:
#    Variable bitsig:  '0'
#    Variable intsig:  24
#    Variable realsig:  20.85
#    Variable timesig:  [0,8]
#    Variable stdlogicsig:  '0'
#    Variable bitarr:  '1'  '0'  '0'  '1'
#    Variable stdlogicarr:  '1'  '0'  '1'  '0'
#    Variable intarr:  13  14  15
#    Variable realarr:  13.1  102.72
#    Variable timearr:  [0,18]  [0,9]
#    Variable rec:
#        '1'
#        4
#        11.2
#        '0'
#        '0'  '1'  '1'  '0'
VSIM 2> quit
```

mti_GetWlfFilename()

Gets the name of the waveform logfile (.wlf).

Syntax

filename = mti_GetWlfFilename()

Arguments

None

Return Values

Name	Type	Description
filename	char *	A pointer to the name of the waveform logfile (.wlf)

Description

mti_GetWlfFilename() returns the name of the waveform logfile (*.wlf*). You must not free the returned pointer.

Examples

FLI code

```
#include <mti.h>

void initForeign(
  mtiRegionIdT        region,    /* The ID of the region in which this     */
                                 /* foreign architecture is instantiated.  */
  char               *param,     /* The last part of the string in the     */
                                 /* foreign attribute.                     */
  mtiInterfaceListT *generics,   /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports       /* A list of ports for the foreign model.  */
)
{
  char * filename;

  filename = mti_GetWlfFilename();
  mti_PrintFormatted( "WLF filename = %s\n", filename );
}
```

HDL code

```
entity top is
end top;

architecture a of top is

   signal s1 : bit := '0';

begin

   s1 <= not s1 after 10 ns;

end a;
```

Simulation output

```
% vsim -c -wlf mydata.wlf top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.5

# vsim -foreign {initForeign for_model.sl} -c -wlf mydata.wlf top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading ./for_model.sl
# WLF filename = mydata.wlf
VSIM 1> add log -r /*
VSIM 2> run 100
VSIM 3> quit
```

mti_HigherRegion()

Gets the parent region of a region.

Syntax

parent_id = mti_HigherRegion(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
parent_id	mtiRegionIdT	A handle to the parent region of the specified region

Description

mti_HigherRegion() returns a handle to the parent region of the specified region or NULL if the specified region is a top-level region. The specified and returned region IDs can be handles to either VHDL, Verilog, or SystemC regions. You can use a handle to a Verilog region with PLI functions to obtain information about or access objects in the Verilog region.

Examples

FLI code

```
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT parent;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s", indent, ' ', region_name );
  mti_VsimFree( region_name );
  parent = mti_HigherRegion( region );
  if ( parent ) {
    mti_PrintFormatted( "   (Parent region is %s)\n",
                        mti_GetRegionName( parent ));
  } else {
    mti_PrintFormatted( "\n" );
  }
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nLoad Done phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this   */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the   */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
  mti_PrintMessage( "\nElaboration phase:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
  end inv;

architecture b of inv is

begin
  b <= a after delay;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;
```

```
entity top is
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Elaboration phase:
#  Region /top
#    Region /top/inst1    (Parent region is top)
#      Region /top/inst1/i1   (Parent region is inst1)
#      Region /top/inst1/flip   (Parent region is inst1)
#
# Load Done phase:
#  Region /top
#    Region /top/inst1    (Parent region is top)
#      Region /top/inst1/flip   (Parent region is inst1)
#      Region /top/inst1/i1   (Parent region is inst1)
#      Region /top/inst1/toggle   (Parent region is inst1)
VSIM 1> quit
```

mti_Image()

Gets the string image of a value of a specific type.

Syntax

strval = mti_Image(value, type_id)

Arguments

Name	Type	Description
value	void *	A pointer to a value that is in the correct format for the specified type
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
strval	char *	A string image of the specified value

Description

mti_Image() returns a pointer to a buffer containing the string image of the specified value. The format is determined by the specified type. The image is the same as would be returned by the VHDL attribute 'IMAGE. The returned string is valid only until the next call to mti_Image(). You must not free the returned pointer.

Examples

FLI code

```c
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiTypeIdT             typeid;
  mtiVariableIdT         varid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;     /* List of variables.*/
  mtiProcessIdT   proc;         /* Test process id. */
} instanceInfoT;

static void checkValues( void *inst_info )
{
  instanceInfoT * inst_data = (instanceInfoT *)inst_info;
  varInfoT      * varinfo;
  void          * value;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );
  for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
    value = mti_GetVarValueIndirect( varinfo->varid, 0 );
    mti_PrintFormatted( "  Variable %s = %s\n",
                        varinfo->name,
                        mti_Image( value, varinfo->typeid ));
  }
  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static varInfoT * setupVariable( mtiVariableIdT varid )
{
  varInfoT * varinfo;

  varinfo          = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
  varinfo->varid   = varid;
  varinfo->name    = mti_GetVarName( varid );
  varinfo->typeid  = mti_GetVarType( varid );
  varinfo->next    = 0;
  return( varinfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiProcessIdT   procid;
  mtiRegionIdT    regid;
  mtiVariableIdT  varid;
  varInfoT      * curr_info;
  varInfoT      * varinfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->var_info = 0;
  regid = mti_GetTopRegion();
```

```
    for ( procid = mti_FirstProcess( regid );
          procid; procid = mti_NextProcess() ) {
      for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
        varinfo = setupVariable( varid );
        if ( inst_data->var_info == 0 ) {
          inst_data->var_info = varinfo;
        }
        else {
          curr_info->next = varinfo;
        }
        curr_info = varinfo;
      }
    }
    inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                         (void *)inst_data );
    mti_ScheduleWakeup( inst_data->proc, 4 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this   */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

end top;

architecture a of top is

  component for_model
  end component;
  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit       := '1';
    variable intsig      : integer   := 21;
    variable realsig     : real      := 16.35;
    variable timesig     : time      := 5 ns;
    variable stdlogicsig : std_logic := 'H';

    variable bitarr      : bitarray  := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray  := ( 10, 11, 12 );
    variable realarr     : realarray := ( 11.6, 101.22 );
    variable timearr     : timearray := ( 15 ns, 6 ns );

  begin

    bitsig      := not bitsig;
    intsig      := intsig + 1;
    realsig     := realsig + 1.1;
    timesig     := timesig + 1 ns;
    stdlogicsig := not stdlogicsig;

    bitarr      := not bitarr;
    stdlogicarr := not stdlogicarr;

    intarr(1)   := intarr(1) + 1;
```

```
        intarr(2)   := intarr(2) + 1;
        intarr(3)   := intarr(3) + 1;

        realarr(1)  := realarr(1) + 1.1;
        realarr(2)  := realarr(2) + 1.1;

        timearr(-1) := timearr(-1) + 1 ns;
        timearr(0)  := timearr(0)  + 1 ns;

        wait for 5 ns;

    end process;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 12
# Time [0,4]:
#   Variable bitsig = '0'
#   Variable intsig = 22
#   Variable realsig = 1.745000e+01
#   Variable timesig = 6 ns
#   Variable stdlogicsig = '0'
#   Variable bitarr = "1001"
#   Variable stdlogicarr = "1010"
#   Variable intarr = (11, 12, 13)
#   Variable realarr = (1.270000e+01, 1.023200e+02)
#   Variable timearr = (16 ns, 7 ns)
# Time [0,9]:
#   Variable bitsig = '1'
#   Variable intsig = 23
#   Variable realsig = 1.855000e+01
#   Variable timesig = 7 ns
#   Variable stdlogicsig = '1'
#   Variable bitarr = "0110"
#   Variable stdlogicarr = "0101"
#   Variable intarr = (12, 13, 14)
#   Variable realarr = (1.380000e+01, 1.034200e+02)
#   Variable timearr = (17 ns, 8 ns)
VSIM 2> quit
```

mti_Interp()

Gets the Tcl_Interp pointer used in the simulator.

Syntax

interp = mti_Interp()

Arguments

None

Return Values

Name	Type	Description
interp	Tcl_Interp *	The Tcl interp pointer used in the simulator

Description

mti_Interp() returns the Tcl interp pointer used in the simulator. There is only one Tcl interp pointer in the simulator and it exists and does not change throughout the execution life of the simulator. This pointer is needed in most Tcl calls and can also be used in conjunction with mti_Cmd() to obtain the command results.

Examples

FLI code

```
#include <tcl.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,        /* 'U' */
  STD_LOGIC_X,        /* 'X' */
  STD_LOGIC_0,        /* '0' */
  STD_LOGIC_1,        /* '1' */
  STD_LOGIC_Z,        /* 'Z' */
  STD_LOGIC_W,        /* 'W' */
  STD_LOGIC_L,        /* 'L' */
  STD_LOGIC_H,        /* 'H' */
  STD_LOGIC_D         /* '-' */
} StdLogicT;

void monitorSignal( void * param )
{
  char          buffer[256];
  char *        region_name;
  char *        signal_name;
  int           status;
  mtiSignalIdT sigid = (mtiSignalIdT)param;
  Tcl_Interp * interp;

  switch ( mti_GetSignalValue( sigid ) ) {
    case STD_LOGIC_X:
    case STD_LOGIC_W:
      signal_name = mti_GetSignalName( sigid );
      region_name = mti_GetRegionFullName( mti_GetSignalRegion( sigid ));
      mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is UNKNOWN\n",
                          mti_NowUpper(), mti_Now(), mti_Delta(),
                          region_name, signal_name );
      sprintf( buffer, "drivers %s/%s", region_name, signal_name );
      interp = mti_Interp();
      status = mti_Cmd( buffer );
      if ( status != TCL_OK ) {
        mti_PrintMessage( "ERROR while executing drivers command.\n" );
      } else {
        mti_PrintFormatted( "The drivers of %s/%s are:\n%s\n",
                region_name, signal_name, Tcl_GetStringResult(interp) );
      }
      Tcl_ResetResult( interp );
      mti_VsimFree( region_name );
      break;
    default:
      break;
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
```

```
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;

  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "SignalMonitor", monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

begin

  p1 : process
  begin
    c1 : case s1 is
      when 'U' => s1 <= 'X' after 5 ns;
      when 'X' => s1 <= '0' after 5 ns;
      when '0' => s1 <= '1' after 5 ns;
      when '1' => s1 <= 'Z' after 5 ns;
      when 'Z' => s1 <= 'W' after 5 ns;
      when 'W' => s1 <= 'L' after 5 ns;
      when 'L' => s1 <= 'H' after 5 ns;
      when 'H' => s1 <= '-' after 5 ns;
      when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    wait for 5 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 60
# Time [0,15] delta 0: Signal /top/s1 is UNKNOWN
# The drivers of /top/s1 are:
# Drivers for /top/s1:
#  W : Signal /top/s1
#    W : Driver /top/p1
#
# Time [0,40] delta 0: Signal /top/s1 is UNKNOWN
# The drivers of /top/s1 are:
# Drivers for /top/s1:
#  X : Signal /top/s1
#    X : Driver /top/p1
#
# Time [0,60] delta 0: Signal /top/s1 is UNKNOWN
# The drivers of /top/s1 are:
# Drivers for /top/s1:
#  W : Signal /top/s1
#    W : Driver /top/p1
#
VSIM 2> quit
```

mti_IsColdRestore()

Determines if a cold restore operation is in progress.

Syntax

status = mti_IsColdRestore()

Arguments

None

Return Values

Name	Type	Description
status	int	1 when a cold restore operation is in progress; 0 otherwise

Description

mti_IsColdRestore() returns 1 when a cold restore operation is in progress; otherwise, it returns 0. A cold restore is when the simulator has been terminated and is re-invoked with the -restore argument.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instance_info = malloc( strlen(param) + 1 );
  if ( mti_IsColdRestore() ) {
      mti_PrintMessage( "Cold Restore in progress ...\n" );
  } else {
      strcpy( instance_info, param );
  }
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 45
VSIM 2> checkpoint cpfile
# Saving instance info "for_model"
VSIM 3> quit
# Cleaning up...
% vsim -c top -restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Wed Jul  5 11:02:06 2000
# Restoring state at time 45 ns, iteration 1
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Wed Jul  5 11:02:06 2000
# Restoring state at time 45 ns, iteration 1
# Loading ./for_model.sl
# Cold Restore in progress ...
# Restored instance info "for_model"
# Simulation kernel restore completed
# Restoring graphical user interface: definitions of virtuals; contents of
list and wave windows
# env sim:/top
# sim:/top
VSIM 1> quit
# Cleaning up...
```

mti_IsFirstInit()

Detects the first call to the initialization function.

Syntax

status = mti_IsFirstInit()

Arguments

None

Return Values

Name	Type	Description
status	int	1 during the first call to the initialization function; 0 otherwise

Description

mti_IsFirstInit() returns 1 during the first call to the initialization function or 0 if the simulation has been restarted.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;
static int    test_global = 15;

void cleanupCallback( void * param )
{
  mti_PrintMessage( "\nCleanup callback:\n" );
  mti_PrintFormatted( "  Freeing param \"%s\"...\n", param );
  free( param );
  mti_PrintFormatted( "  test_global = %d\n", test_global );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  if ( mti_IsFirstInit() ) {
    mti_PrintMessage( "\nFirst call to init function.\n" );
    mti_PrintFormatted( "  test_global = %d\n", test_global );
    test_global = 42;
    mti_PrintFormatted( "  Setting test_global to %d\n", test_global );
  } else {
    mti_PrintMessage( "\nSimulation has been restarted.\n" );
    mti_PrintFormatted( "  test_global = %d\n", test_global );
    test_global = 3;
    mti_PrintFormatted( "  Setting test_global to %d\n", test_global );
  }
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# First call to init function.
#   test_global = 15
#   Setting test_global to 42
VSIM 1> run 30
VSIM 2> restart -f
#
# Cleanup callback:
#   Freeing param "for_model"...
#   test_global = 42
# Loading ./for_model.sl
#
# Simulation has been restarted.
#   test_global = 15
#   Setting test_global to 3
VSIM 3> run 45
VSIM 4> quit
#
# Cleanup callback:
#   Freeing param "for_model"...
#   test_global = 3
```

mti_IsRestore()

Determines if a restore operation is in progress.

Syntax

status = mti_IsRestore()

Arguments

None

Return Values

Name	Type	Description
status	int	1 during a restore operation; 0 otherwise

Description

mti_IsRestore() returns 1 when a restore operation is in progress; otherwise, it returns 0.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info = 0;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  if ( ! instance_info ) {
      instance_info = malloc( strlen(param) + 1 );
  }
  if ( mti_IsRestore() ) {
      mti_PrintMessage( "Restore in progress ...\n" );
  } else {
      strcpy( instance_info, param );
  }
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 40
VSIM 2> checkpoint cpfile
# Saving instance info "for_model"
VSIM 3> run 30
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Wed Jul  5 14:04:26 2000
# Restoring state at time 40 ns, iteration 1
# Restore in progress ...
# Restored instance info "for_model"
VSIM 5> echo $now
# 40
VSIM 6> run 10
VSIM 7> quit
# Cleaning up...
% vsim -c top -restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Wed Jul  5 14:04:26 2000
# Restoring state at time 40 ns, iteration 1
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Wed Jul  5 14:04:26 2000
# Restoring state at time 40 ns, iteration 1
# Loading ./for_model.sl
# Restore in progress ...
# Restored instance info "for_model"
# Simulation kernel restore completed
# Restoring graphical user interface: definitions of virtuals; contents of
list and wave windows
# env sim:/top
# sim:/top
VSIM 1> run 25
VSIM 2> quit
# Cleaning up...
```

mti_IsSystemcType()

Determines if the argument is a handle to a SystemC type.

Syntax

status = mti_IsSystemcType(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
status	int	1 if type_id is a handle to a SystemC type; 0 otherwise

Description

mti_IsSystemcType() returns 1 if the argument is a handle to a systemC type, 0 otherwise.

mti_IsSystemcSignedType()

Determines if the argument is a handle to a SystemC signed type.

Syntax

status = mti_IsSystemcSignedType(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
status	int	1 if type_id is a handle to a SystemC signed type; 0 otherwise

Description

mti_IsSystemcSignedType() returns 1 if the argument is a handle to a systemC signed type, 0 otherwise.

mti_KeepLoaded()

Requests that the current shared library not be unloaded on restart or load of a new design.

Syntax

mti_KeepLoaded()

Arguments

None

Return Values

Nothing

Description

mti_KeepLoaded() marks the current shared library as not to be reloaded when a restart or load of a new design occurs. You must call mti_KeepLoaded() from the initialization function of a foreign architecture.

Normally, the reloading of shared libraries is determined by the following:

- Loading a shared library due to reloading a foreign attribute on a VHDL architecture.

- Loading a shared library loaded due to reloading the -foreign option to vsim.

- Loading a shared library due to not reloading a foreign attribute on a VHDL subprogram, even if the shared library also contains code for a foreign architecture.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;
static int    test_global = 15;

void cleanupCallback( void * param )
{
  mti_PrintMessage( "\nCleanup callback:\n" );
  mti_PrintFormatted( "  Freeing param \"%s\"\n", param );
  free( param );
  mti_PrintFormatted( "  test_global = %d\n", test_global );
}

void restartCallback( void * param )
{
  mti_PrintMessage( "\nRestart callback:\n" );
  mti_PrintFormatted( "  Param is \"%s\"\n", param );
  mti_PrintFormatted( "  test_global = %d\n", test_global );
  test_global = 15;
  mti_PrintFormatted( "  Setting test_global to initial value of %d\n",
                      test_global );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  if ( mti_IsFirstInit() ) {
    mti_PrintMessage( "\nFirst call to init function.\n" );
    mti_PrintFormatted( "  test_global = %d\n", test_global );
    test_global = 42;
    mti_PrintFormatted( "  Setting test_global to %d\n", test_global );
    mti_PrintFormatted( "  Shared library will NOT be reloaded.\n" );
    mti_KeepLoaded();
    instance_info = malloc( strlen(param) + 1 );
    strcpy( instance_info, param );
  } else {
    mti_PrintMessage( "\nSimulation has been restarted.\n" );
    mti_PrintFormatted( "  test_global = %d\n", test_global );
    test_global = 3;
    mti_PrintFormatted( "  Setting test_global to %d\n", test_global );
  }
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( restartCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# First call to init function.
#   test_global = 15
#   Setting test_global to 42
#   Shared library will NOT be reloaded.
VSIM 1> run 30
VSIM 2> restart -f
#
# Restart callback:
#   Param is "for_model"
#   test_global = 42
#   Setting test_global to initial value of 15
#
# Simulation has been restarted.
#   test_global = 15
#   Setting test_global to 3
VSIM 3> run 100
VSIM 4> restart -f
#
# Restart callback:
#   Param is "for_model"
#   test_global = 3
#   Setting test_global to initial value of 15
#
# Simulation has been restarted.
#   test_global = 15
#   Setting test_global to 3
VSIM 5> quit
#
# Cleanup callback:
#   Freeing param "for_model"
#   test_global = 3
```

mti_Malloc()

Allocates simulator-managed memory.

Syntax

memptr = mti_Malloc(size)

Arguments

Name	Type	Description
size	unsigned long	The size in bytes of the memory to be allocated

Return Values

Name	Type	Description
memptr	void *	A pointer to the allocated memory

Description

mti_Malloc() allocates a block of memory of the specified size from an internal simulator memory pool and returns a pointer to it. The simulator initializes the memory to zero, and automatically checkpoints memory allocated by mti_Malloc(). On restore, this memory is guaranteed to be restored to the same location with the values it contained at the time of the checkpoint. You can free this memory only by mti_Free().

mti_Malloc() automatically checks for a NULL pointer. In the case of an allocation error, mti_Malloc() issues the following error message and aborts the simulation:

```
****** Memory allocation failure. *****
Please check your system for available memory and swap space.
```

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info pointer to \"%s\"\n",
                      inst_info );
  mti_SaveBlock( (char *)&inst_info, sizeof(inst_info) );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  /* NOTE:  Memory allocated by mti_Malloc() will be freed by vsim.  */
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  if ( mti_IsRestore() ) {
    mti_PrintMessage( "Restore in progress ...\n" );
  } else {
    instance_info = mti_Malloc( strlen(param) + 1 );
    strcpy( instance_info, param );
  }
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
   attribute foreign of a : architecture is
      "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

   signal s1 : bit := '0';

   component for_model is
   end component;

   for all : for_model use entity work.for_model(a);

begin

   i1 : for_model;

   s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 35
VSIM 2> checkpoint cpfile
# Saving instance info pointer to "my_for_model"
VSIM 3> run 10
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Wed Jul  5 15:24:18 2000
# Restoring state at time 35 ns, iteration 1
# Restore in progress ...
# Restored instance info "my_for_model"
VSIM 5> run 20
VSIM 6> quit
# Cleaning up...
```

mti_NextProcess()

Gets the next process in a region.

Syntax

process_id = mti_NextProcess()

Arguments

None

Return Values

Name	Type	Description
process_id	mtiProcessIdT	A handle to the next VHDL or SystemC process in the current region

Description

mti_NextProcess() returns a handle to the next process in the region set by the latest call to mti_FirstProcess(). mti_NextProcess() returns NULL if there are no more processes.

Examples

FLI code

```c
#include <mti.h>

void printProcesses( mtiRegionIdT region, int indent )
{
  mtiProcessIdT procid;

  for ( procid = mti_FirstProcess( region ); procid;
        procid = mti_NextProcess() ) {
    if ( procid ) {
      mti_PrintFormatted( "%*cProcess %s\n", indent, ' ',
                          mti_GetProcessName( procid ) );
    }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printProcesses( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nHierarchy:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;

  p1 : process
    variable count : integer := 0;
  begin
    count := count + 1;
    wait on a;
  end process;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

component inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
```

```
   s3 <= not s3 after 5 ns;

   toggle : inv port map ( s1, s2 );

end a;

entity top is
end top;

architecture a of top is
  component mid is
end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Hierarchy:
#  Region /top
#    Region /top/inst1
#       Process line__58
#       Process line__57
#      Region /top/inst1/flip
#         Process p1
#         Process line__19
#      Region /top/inst1/i1
#      Region /top/inst1/toggle
#         Process p1
#         Process line__19
VSIM 1> quit
```

mti_NextRegion()

Gets the next region at the same level as a region.

Syntax

next_reg_id = mti_NextRegion(region_id)

Arguments

Name	Type	Description
region_id	mtiRegionIdT	A handle to a VHDL, Verilog, or SystemC region

Return Values

Name	Type	Description
next_reg_id	mtiRegionIdT	A handle to the next VHDL, Verilog or SystemC region at the same level of hierarchy as the specified region

Description

mti_NextRegion() returns a handle to the next VHDL, Verilog, or SystemC region at the same level of hierarchy as the specified VHDL, Verilog, or SystemC region. mti_NextRegion() returns NULL if there are no more regions at this level. If the next_reg_id is a handle to a Verilog region then you can use it with PLI functions to obtain information about or access objects in the Verilog region.

Examples

FLI code

```
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  mti_VsimFree( region_name );
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mtiRegionIdT regid;

  mti_PrintMessage( "\nHierarchy:\n" );
  for ( regid = mti_GetTopRegion();
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, 1 );
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
```

```
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Hierarchy:
#  Region /top
#    Region /top/inst1
#      Region /top/inst1/flip
#      Region /top/inst1/i1
#      Region /top/inst1/toggle
#  Region /standard
VSIM 1> quit
```

mti_NextSignal()

Gets the next VHDL or SystemC signal in a region.

Syntax

signal_id = mti_NextSignal()

Arguments

None

Return Values

Name	Type	Description
signal_id	mtiSignalIdT	A handle to the next VHDL or SystemC signal in the current region

Description

mti_NextSignal() returns a handle to the next signal in the region set by the latest call to mti_FirstSignal(). mti_NextSignal() returns NULL if there are no more signals.

Examples

FLI code

```
#include <mti.h>

void printSignals( mtiRegionIdT region, int indent )
{
  mtiSignalIdT sigid;

  for ( sigid = mti_FirstSignal( region ); sigid;
        sigid = mti_NextSignal() ) {
    mti_PrintFormatted( "%*cSignal %s\n", indent, ' ',
                        mti_GetSignalName( sigid ) );
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  mti_VsimFree( region_name );
  indent += 2;
  printSignals( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nHierarchy:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this   */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the   */
                              /* foreign attribute.                   */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
  signal count : integer := 0;
begin
  b <= a after delay;

  p1 : process( a )
  begin
    count <= count + 1 after 0 ns;
  end process;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;
```

```
    toggle : inv port map ( s1, s2 );

end a;

entity top is
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Hierarchy:
#  Region /top
#    Region /top/inst1
#       Signal s1
#       Signal s2
#       Signal s3
#       Signal s4
#       Region /top/inst1/flip
#          Signal a
#          Signal b
#          Signal count
#       Region /top/inst1/i1
#       Region /top/inst1/toggle
#          Signal a
#          Signal b
#          Signal count
VSIM 1> quit
```

mti_NextVar()

Gets the next VHDL variable, generic, or constant, or System C variable visible to a process.

Syntax

variable_id = mti_NextVar()

Arguments

None

Return Values

Name	Type	Description
variable_id	mtiVariableIdT	A handle to the next VHDL variable, generic, or constant, or SystemC variable visible to the current process

Description

mti_NextVar() returns a handle to the next variable, generic, or constant visible to the process set by the latest call to mti_FirstVar(). mti_NextVar() returns NULL if there are no more variables, generics, or constants.

Examples

FLI code

```
#include <mti.h>

void printVariables( mtiProcessIdT process, int indent )
{
  mtiVariableIdT varid;

  for ( varid = mti_FirstVar( process ); varid; varid = mti_NextVar() ) {
    if ( varid ) {
      mti_PrintFormatted( "%*cVariable %s\n", indent, ' ',
                          mti_GetVarName( varid ) );
    }
  }
}

void printProcesses( mtiRegionIdT region, int indent )
{
  mtiProcessIdT procid;

  for ( procid = mti_FirstProcess( region ); procid;
        procid = mti_NextProcess() ) {
    if ( procid ) {
      mti_PrintFormatted( "%*cProcess %s\n", indent, ' ',
                          mti_GetProcessName( procid ) );
      printVariables( procid, indent+2 );
    }
  }
}

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s\n", indent, ' ', region_name );
  indent += 2;
  printProcesses( region, indent );
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
  mti_VsimFree( region_name );
}

void loadDoneCB( void * param )
{
  mti_PrintMessage( "\nHierarchy:\n" );
  printHierarchy( mti_GetTopRegion(), 1 );
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this    */
                                /* foreign architecture is instantiated. */
  char               *param,    /* The last part of the string in the    */
```

```
                                    /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

  architecture b of inv is
begin
  b <= a after delay;

  p1 : process
    constant increment : integer := 1;
    variable count : integer := 0;
  begin
    count := count + increment;
    wait on a;
  end process;
end b;

entity mid is
  generic ( gen1 : string := "Mid" );
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  testproc : process
    constant c1 : string := "mystring";
    variable v1 : bit := '0';
```

```
      variable v2 : integer := 42;
      variable v3 : real := 7.82;
  begin
    v1 := not v1;
    v2 := v2 + 2;
    v3 := v3 + 1.5;
    wait for 5 ns;
  end process;

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
end top;

architecture a of top is
  component mid is
    generic ( gen1 : string := "Top" );
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Hierarchy:
#   Region /top
#     Region /top/inst1
#       Process line__72
#         Variable gen1
#       Process line__71
#         Variable gen1
#       Process testproc
#         Variable gen1
#         Variable c1
#         Variable v1
#         Variable v2
#         Variable v3
#       Region /top/inst1/flip
#         Process p1
#           Variable delay
#           Variable increment
#           Variable count
#         Process line__19
#           Variable delay
#       Region /top/inst1/i1
#       Region /top/inst1/toggle
#         Process p1
#           Variable delay
#           Variable increment
#           Variable count
#         Process line__19
#           Variable delay
VSIM 1> quit
```

mti_Now()

Gets the low order 32 bits of the 64-bit current simulation time.

Syntax

low_time = mti_Now()

Arguments

None

Return Values

Name	Type	Description
low_time	mtiInt32T	The low order 32 bits of the current simulation time

Description

mti_Now() returns the low order 32 bits of the current simulation time. The time units are equivalent to the current simulator time unit setting.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;     /* List of signals. */
  mtiProcessIdT  proc;         /* Test process id.*/
} instanceInfoT;

static char * convertTime( mtiInt32T time, int limit, mtiInt32T * new_time )
{
  switch ( limit ) {
    case 2:    *new_time = time * 100;  return( "sec" );
    case 1:    *new_time = time * 10;   return( "sec" );
    case 0:    *new_time = time;        return( "sec" );
    case -1:   *new_time = time * 100;  return( "ms" );
    case -2:   *new_time = time * 10;   return( "ms" );
    case -3:   *new_time = time;        return( "ms" );
    case -4:   *new_time = time * 100;  return( "us" );
    case -5:   *new_time = time * 10;   return( "us" );
    case -6:   *new_time = time;        return( "us" );
    case -7:   *new_time = time * 100;  return( "ns" );
    case -8:   *new_time = time * 10;   return( "ns" );
    case -9:   *new_time = time;        return( "ns" );
    case -10:  *new_time = time * 100;  return( "ps" );
    case -11:  *new_time = time * 10;   return( "ps" );
    case -12:  *new_time = time;        return( "ps" );
    case -13:  *new_time = time * 100;  return( "fs" );
    case -14:  *new_time = time * 10;   return( "fs" );
    case -15:  *new_time = time;        return( "fs" );
    default:   *new_time = time;        return( "??" );
  }
}

static void checkValues( void *inst_info )
{
  char          * units;
  instanceInfoT * inst_data = (instanceInfoT *)inst_info;
  mtiInt32T       new_time;
  signalInfoT   * siginfo;

  units = convertTime( mti_Now(), mti_GetResolutionLimit(), &new_time );
  mti_PrintFormatted( "Time %d %s:\n", new_time, units );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
    mti_PrintFormatted( "  Signal %s: %s\n",
                        siginfo->name, mti_SignalImage( siginfo->sigid ));
  }
```

```
    mti_ScheduleWakeup( inst_data->proc, 5 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid  = sigid;
  siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid = mti_GetSignalType( sigid );
  siginfo->next   = 0;

  return( siginfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    siginfo = setupSignal( sigid );
    if ( inst_data->sig_info == 0 ) {
      inst_data->sig_info = siginfo;
    }
    else {
      curr_info->next = siginfo;
    }
    curr_info = siginfo;
  }

  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 6 );
}
void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal bitsig      : bit        := '1';
  signal intsig      : integer    := 42;
  signal realsig     : real       := 10.2;
  signal timesig     : time       := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig      <= not bitsig after 5 ns;
  intsig      <= intsig + 1 after 5 ns;
  realsig     <= realsig + 1.1 after 5 ns;
  timesig     <= timesig + 2 ns after 5 ns;
  stdlogicsig <= not stdlogicsig after 5 ns;
  stdlogicarr <= not stdlogicarr after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 17
# Time 6 ns:
#   Signal bitsig: '0'
#   Signal intsig: 43
#   Signal realsig: 1.130000e+01
#   Signal timesig: 5 ns
#   Signal stdlogicsig: '0'
#   Signal stdlogicarr: "1010"
# Time 11 ns:
#   Signal bitsig: '1'
#   Signal intsig: 44
#   Signal realsig: 1.240000e+01
#   Signal timesig: 7 ns
#   Signal stdlogicsig: '1'
#   Signal stdlogicarr: "0101"
# Time 16 ns:
#   Signal bitsig: '0'
#   Signal intsig: 45
#   Signal realsig: 1.350000e+01
#   Signal timesig: 9 ns
#   Signal stdlogicsig: '0'
#   Signal stdlogicarr: "1010"
VSIM 2> quit
```

mti_NowFormatted()

Returns the current simulation time formatted according to specified flags.

Syntax

value = mti_NowFormatted(flags)

Arguments

Name	Type	Description
flags	mtiTimeFlagT	The type of formatting for the current simulation time.

Return Values

Name	Type	Description
value	char *	Returns the current simulation time formatted based on your settings.

Description

mti_NowFormatted() returns the current simulation time using the settings as specified in the function. The simulation stores the return value in a buffer and you should use it immediately.

You can specify any number of flag arguments in a pipe (|) separated list.

The argument *flag* can include any of the following:

MTI_TIME_BEST_UNITS	Determines the best unit to use for display.
MTI_TIME_INSERT_COMMAS	Inserts commas every three digits.
MTI_TIME_NO_DEF_UNIT	Does not display the default units.
MTI_TIME_FREQUENCY	Displays the time as 1/*time* in hz.

mti_NowIndirect()

Gets the upper and lower 32 bits of the 64-bit current simulation time.

Syntax

curr_time = mti_NowIndirect(time_buf)

Arguments

Name	Type	Description
time_buf	mtiTime64T *	Returns the upper and lower 32 bits of the current simulation time; OPTIONAL - can be NULL

Return Values

Name	Type	Description
curr_time	mtiTime64T *	The upper and lower 32 bits of the current simulation time

Description

mti_NowIndirect() returns the upper and lower 32 bits of the 64-bit current simulation time. The time units are equivalent to the current simulator time unit setting. If the time_buf parameter is NULL, then mti_NowIndirect() allocates memory for the value and returns a pointer to it. The caller is responsible for freeing this memory with mti_VsimFree(). If the time_buf parameter is not NULL, then mti_NowIndirect() copies the value into the time_buf parameter and also returns the time_buf parameter.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                    * name;
  mtiSignalIdT              sigid;
  mtiTypeIdT                typeid;
} signalInfoT;

typedef struct {
  signalInfoT  * sig_info;      /* List of signals.*/
  mtiProcessIdT   proc;         /* Test process id. */
} instanceInfoT;

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  mtiTime64T      curr_time;
  signalInfoT    *siginfo;

  (void) mti_NowIndirect( &curr_time );
  mti_PrintFormatted( "Time [%d,%d]:\n",
                      MTI_TIME64_HI32( curr_time ),
                      MTI_TIME64_LO32( curr_time ));

for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
  mti_PrintFormatted( "  Signal %s: %s\n",
                      siginfo->name, mti_SignalImage( siginfo->sigid ));
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo          = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid   = sigid;
  siginfo->name    = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid  = mti_GetSignalType( sigid );
  siginfo->next    = 0;

  return( siginfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
```

```
      inst_data->sig_info = 0;

      for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
            sigid; sigid = mti_NextSignal() ) {
        siginfo = setupSignal( sigid );
        if ( inst_data->sig_info == 0 ) {
          inst_data->sig_info = siginfo;
        }
        else {
          curr_info->next = siginfo;
        }
        curr_info = siginfo;
      }

      inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                           (void *)inst_data );
      mti_ScheduleWakeup( inst_data->proc, 6 );
    }

    void initForeign(
      mtiRegionIdT        region,   /* The ID of the region in which this    */
                                    /* foreign architecture is instantiated. */
      char               *param,    /* The last part of the string in the    */
                                    /* foreign attribute.                    */
      mtiInterfaceListT *generics,  /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports      /* A list of ports for the foreign model.   */
    )
    {
      mti_AddLoadDoneCB( initInstance, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig       : bit        := '1';
  signal intsig       : integer    := 42;
  signal physsig      : bigtime    := 3 hour;
  signal realsig      : real       := 10.2;
  signal timesig      : time       := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

  signal rec          : rectype    := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig       <= not bitsig after 5 ns;
  intsig       <= intsig + 1 after 5 ns;
```

```
    physsig     <= physsig + 1 hour after 5 ns;
    realsig     <= realsig + 1.1 after 5 ns;
    timesig     <= timesig + 2 ns after 5 ns;
    stdlogicsig <= not stdlogicsig after 5 ns;

    stdlogicarr <= not stdlogicarr after 5 ns;

    rec.a       <= not rec.a after 5 ns;
    rec.b       <= rec.b + 1 after 5 ns;
    rec.c       <= not rec.c after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 14
# Time [0,6]:
#   Signal bitsig: '0'
#   Signal intsig: 43
#   Signal physsig: 4 hour
#   Signal realsig: 1.130000e+01
#   Signal timesig: 5 ns
#   Signal stdlogicsig: '0'
#   Signal stdlogicarr: "1010"
#   Signal rec: ('1', 1, "0110")
# Time [0,11]:
#   Signal bitsig: '1'
#   Signal intsig: 44
#   Signal physsig: 5 hour
#   Signal realsig: 1.240000e+01
#   Signal timesig: 7 ns
#   Signal stdlogicsig: '1'
#   Signal stdlogicarr: "0101"
#   Signal rec: ('0', 2, "1001")
VSIM 2> quit
```

mti_NowUpper()

Gets the high order 32 bits of the 64-bit current simulation time.

Syntax

high_time = mti_NowUpper()

Arguments

None

Return Values

Name	Type	Description
high_time	mtiInt32T	The high order 32 bits of the current simulation time

Description

mti_NowUpper() returns the high order 32 bits of the current simulation time. The time units are equivalent to the current simulator time unit setting.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT   * sig_info;     /* List of signals. */
  mtiProcessIdT   proc;         /* Test process id.*/
} instanceInfoT;

static void checkValues( void *inst_info )
{
  instanceInfoT * inst_data = (instanceInfoT *)inst_info;
  signalInfoT   * siginfo;

  mti_PrintFormatted( "Time [%d,%u]:\n", mti_NowUpper(), mti_Now() );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
    mti_PrintFormatted( "  Signal %s: %s\n",
                        siginfo->name, mti_SignalImage( siginfo->sigid ));
  }

  mti_ScheduleWakeup( inst_data->proc, 500000000 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid  = sigid;
  siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid = mti_GetSignalType( sigid );
  siginfo->next   = 0;

  return( siginfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
```

```
      siginfo = setupSignal( sigid );
      if ( inst_data->sig_info == 0 ) {
        inst_data->sig_info = siginfo;
      }
      else {
        curr_info->next = siginfo;
      }
      curr_info = siginfo;
    }

    inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                    (void *)inst_data );
    mti_ScheduleWakeup( inst_data->proc, 600000000 );
}


void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal intsig     : integer   := 42;
  signal realsig    : real      := 10.2;

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  intsig      <= intsig + 1 after 5000000 sec;
  realsig     <= realsig + 1.1 after 5000000 sec;

end a;
```

Simulation output

```
% vsim -c -t sec top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c -t sec top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 10000000000
# Time [0,600000000]:
#   Signal intsig: 162
#   Signal realsig: 1.422000e+02
# Time [0,1100000000]:
#   Signal intsig: 262
#   Signal realsig: 2.522000e+02
# Time [0,1600000000]:
#   Signal intsig: 362
#   Signal realsig: 3.622000e+02
# Time [0,2100000000]:
#   Signal intsig: 462
#   Signal realsig: 4.722000e+02
# Time [0,2600000000]:
#   Signal intsig: 562
#   Signal realsig: 5.822000e+02
# Time [0,3100000000]:
#   Signal intsig: 662
#   Signal realsig: 6.922000e+02
# Time [0,3600000000]:
#   Signal intsig: 762
#   Signal realsig: 8.022000e+02
# Time [0,4100000000]:
#   Signal intsig: 862
#   Signal realsig: 9.122000e+02
# Time [1,305032704]:
#   Signal intsig: 962
#   Signal realsig: 1.022200e+03
# Time [1,805032704]:
#   Signal intsig: 1062
#   Signal realsig: 1.132200e+03
# Time [1,1305032704]:
#   Signal intsig: 1162
#   Signal realsig: 1.242200e+03
# Time [1,1805032704]:
#   Signal intsig: 1262
#   Signal realsig: 1.352200e+03
# Time [1,2305032704]:
#   Signal intsig: 1362
#   Signal realsig: 1.462200e+03
# Time [1,2805032704]:
#   Signal intsig: 1462
#   Signal realsig: 1.572200e+03
# Time [1,3305032704]:
#   Signal intsig: 1562
```

```
#    Signal realsig: 1.682200e+03
# Time [1,3805032704]:
#    Signal intsig: 1662
#    Signal realsig: 1.792200e+03
# Time [2,10065408]:
#    Signal intsig: 1762
#    Signal realsig: 1.902200e+03
# Time [2,510065408]:
#    Signal intsig: 1862
#    Signal realsig: 2.012200e+03
# Time [2,1010065408]:
#    Signal intsig: 1962
#    Signal realsig: 2.122200e+03
VSIM 2> quit
```

mti_PrintFormatted()

Prints a formatted message to the Main window transcript.

Syntax

mti_PrintFormatted(format, ...)

Arguments

Name	Type	Description
format	char *	The formatted string to be printed
...		Zero or more arguments corresponding to the conversion characters in the format string

Return Values

Nothing

Description

mti_PrintFormatted() prints a formatted message in the Main simulator window and in the transcript file. The functionality is similar to the C printf() function. The format string must contain newline characters where line breaks are desired.

Examples

FLI code

```c
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *       region_name;
  mtiRegionIdT parent;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintFormatted( "%*cRegion %s", indent, ' ', region_name );
  mti_VsimFree( region_name );
  parent = mti_HigherRegion( region );
  if ( parent ) {
    mti_PrintFormatted( "   (Parent region is %s)\n",
                        mti_GetRegionName( parent ));
  } else {
    mti_PrintFormatted( "\n" );
  }
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mtiRegionIdT regid;

  mti_PrintFormatted( "\nHierarchy:\n" );
  for ( regid = mti_GetTopRegion();
       regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, 1 );
  }
}

void initForeign(
  mtiRegionIdT      region,    /* The ID of the region in which this   */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                   */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

entity mid is
end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
```

```
    end top;

    architecture a of top is
      component mid is
      end component;
    begin
      inst1 : mid;
    end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading work.top(a)
    # Loading work.mid(a)
    # Loading work.inv(b)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    #
    # Hierarchy:
    #  Region /top
    #    Region /top/inst1   (Parent region is top)
    #      Region /top/inst1/flip   (Parent region is inst1)
    #      Region /top/inst1/i1   (Parent region is inst1)
    #      Region /top/inst1/toggle   (Parent region is inst1)
    #  Region /standard
    VSIM 1> quit
```

mti_PrintMessage()

Prints a message to the Main window transcript.

Syntax

mti_PrintMessage(message)

Arguments

Name	Type	Description
message	char *	The message to be printed

Return Values

Nothing

Description

mti_PrintMessage() prints a message in the Main simulator window and in the transcript file. You can include one or more newline characters in the message string; however, a newline character is provided at the end of the message by default.

Examples

FLI code

```
#include <mti.h>

void printHierarchy( mtiRegionIdT region, int indent )
{
  char *        region_name;
  mtiRegionIdT parent;
  mtiRegionIdT regid;

  region_name = mti_GetRegionFullName( region );
  mti_PrintMessage( region_name );
  mti_VsimFree( region_name );
  parent = mti_HigherRegion( region );
  indent += 2;
  for ( regid = mti_FirstLowerRegion( region );
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, indent );
  }
}

void loadDoneCB( void * param )
{
  mtiRegionIdT regid;

  mti_PrintMessage( "\nHierarchy:" );
  for ( regid = mti_GetTopRegion();
        regid; regid = mti_NextRegion( regid ) ) {
    printHierarchy( regid, 1 );
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCB, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity inv is
  generic ( delay : time := 5 ns );
  port ( a : in bit;
         b : out bit
       );
end inv;

architecture b of inv is
begin
  b <= a after delay;
end b;

  entity mid is
  end mid;

architecture a of mid is

  signal s1 : bit := '0';
  signal s2 : bit := '0';
  signal s3 : bit := '0';
  signal s4 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component inv is
    generic ( delay : time := 5 ns );
    port ( a : in bit;
           b : out bit
         );
  end component;

begin

  flip : inv port map ( s3, s4 );

  i1 : for_model;

  s1 <= not s1 after 5 ns;
  s3 <= not s3 after 5 ns;

  toggle : inv port map ( s1, s2 );

end a;

entity top is
```

```
end top;

architecture a of top is
  component mid is
  end component;
begin
  inst1 : mid;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.mid(a)
# Loading work.inv(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
#
# Hierarchy:
# /top
# /top/inst1
# /top/inst1/flip
# /top/inst1/i1
# /top/inst1/toggle
# /standard
VSIM 1> quit
```

mti_Quit()

Requests the simulator to exit immediately.

Syntax

mti_Quit()

Arguments

None

Return Values

Nothing

Description

mti_Quit() shuts down the simulator immediately.

Examples

FLI code

```
#include <mti.h>

typedef enum {
  STD_LOGIC_U,        /* 'U' */
  STD_LOGIC_X,        /* 'X' */
  STD_LOGIC_0,        /* '0' */
  STD_LOGIC_1,        /* '1' */
  STD_LOGIC_Z,        /* 'Z' */
  STD_LOGIC_W,        /* 'W' */
  STD_LOGIC_L,        /* 'L' */
  STD_LOGIC_H,        /* 'H' */
  STD_LOGIC_D         /* '-' */
} StdLogicT;

void monitorSignal( void * param )
{
  mtiSignalIdT sigid = (mtiSignalIdT)param;

  switch ( mti_GetSignalValue( sigid ) ) {
    case STD_LOGIC_X:
    case STD_LOGIC_W:
      mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s is UNKNOWN\n",
                          mti_NowUpper(), mti_Now(), mti_Delta(),
                          mti_GetSignalName( sigid ) );
      mti_Quit();
      break;
    default:
      break;
  }
}

void initForeign(
  mtiRegionIdT        region,   /* The ID of the region in which this     */
                                /* foreign architecture is instantiated.  */
  char               *param,    /* The last part of the string in the     */
                                /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;

  sigid  = mti_FindSignal( "/top/s1" );
  procid = mti_CreateProcess( "SignalMonitor", monitorSignal, sigid );
  mti_Sensitize( procid, sigid, MTI_EVENT );
}
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

begin

  p1 : process
  begin
    c1 : case s1 is
      when 'U' => s1 <= 'X' after 5 ns;
      when 'X' => s1 <= '0' after 5 ns;
      when '0' => s1 <= '1' after 5 ns;
      when '1' => s1 <= 'Z' after 5 ns;
      when 'Z' => s1 <= 'W' after 5 ns;
      when 'W' => s1 <= 'L' after 5 ns;
      when 'L' => s1 <= 'H' after 5 ns;
      when 'H' => s1 <= '-' after 5 ns;
      when '-' => s1 <= 'U' after 5 ns;
    end case c1;
    wait for 5 ns;
  end process;

end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading ./for_model.sl
VSIM 1> run 30
# Time [0,15] delta 0: Signal s1 is UNKNOWN
%
```

mti_QuitWithErrorCode()

Allows you to specify a nominal filename and line number along with an exit code, which is reported before terminating the simulator.

Syntax

mti_QuitWithErrorCode(file_name, line_number, error_code)

Arguments

Name	Type	Description
file_name	char *	Name of a file
line_number	int	Line number of the file
error_code	int	Return status of the program

Return Values

Nothing

Description

This issues an error message, with the number 10003, reporting the file_name as a string, and the line_number and error_code as integers. The simulator will then immediately terminate with the status specified by error_code.

mti_Realloc()

Reallocates simulator-managed memory.

Syntax

memptr = mti_Realloc(origptr, size)

Arguments

Name	Type	Description
origptr	void *	A pointer to the currently allocated memory
size	unsigned long	The size in bytes of the new memory to be allocated

Return Values

Name	Type	Description
memptr	void *	A pointer to the reallocated memory

Description

mti_Realloc() works like the C realloc() function on memory allocated by mti_Malloc(). If the specified size is larger than the size of memory already allocated to the origptr parameter, then new memory of the required size is allocated and initialized to zero, the entire content of the old memory is copied into the new memory, and a pointer to the new memory is returned. Otherwise, a pointer to the old memory is returned.

Any memory allocated by mti_Realloc() is guaranteed to be checkpointed and restored just like memory allocated by mti_Malloc(). Memory allocated by mti_Realloc() can be freed only by mti_Free().

mti_Realloc() automatically checks for a NULL pointer. In the case of an allocation error, mti_Realloc() issues the following error message and aborts the simulation:

```
****** Memory allocation failure. *****
Please check your system for available memory and swap space
```

Examples

FLI code

```c
#include <stdlib.h>
#include <stdio.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info pointer to \"%s\"\n",
                      inst_info );
  mti_SaveBlock( (char *)&inst_info, sizeof(inst_info) );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  /*
   * NOTE:  Memory allocated by mti_Malloc() and mti_Realloc() will
   *        be freed by vsim.
   */
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  if ( mti_IsRestore() ) {
    mti_PrintMessage( "Restore in progress ...\n" );
  } else {
    instance_info = mti_Malloc( strlen(param) + 1 );
    strcpy( instance_info, param );
    if ( ! mti_IsFirstInit() ) {
      instance_info = mti_Realloc( instance_info, strlen(param) + 9 );
      sprintf( instance_info, "%s_restart", param );
    }
  }
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
# Saving instance info pointer to "my_for_model"
VSIM 3> run 30
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 13:20:28 2000
# Restoring state at time 20 ns, iteration 1
# Restore in progress ...
# Restored instance info "my_for_model"
VSIM 5> run 40
VSIM 6> restart -f
# Cleaning up...
# Loading ./for_model.sl
VSIM 7> run 15
VSIM 8> checkpoint cpf2
# Saving instance info pointer to "my_for_model_restart"
VSIM 9> run 25
VSIM 10> restore cpf2
# Loading checkpoint/restore data from file "cpf2"
# Checkpoint created Fri Jul  7 13:20:52 2000
# Restoring state at time 15 ns, iteration 1
# Restore in progress ...
# Restored instance info "my_for_model_restart"
VSIM 11> run 35
VSIM 12> quit
# Cleaning up...
```

mti_ReleaseSignal()

Releases a force on a VHDL signal.

Syntax

status = mti_ReleaseSignal(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL signal

Return Values

Name	Type	Description
status	int	1 if successful; 0 if there is an error

Description

mti_ReleaseSignal() releases the specified signal from any active force. mti_ReleaseSignal() returns 1 if the release is successful; otherwise, it returns 0.

Examples

FLI code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef WIN32
#include <unistd.h>
#endif

#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  struct signalInfoT_tag * child;
  char                   * name;
  void                   * last_value;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
  mtiTypeKindT             typekind;
  mtiDirectionT            direction;
  char                     granulate;
} signalInfoT;

typedef struct {
  signalInfoT    * sig_info;        /* List of signals.         */
  mtiProcessIdT    proc;            /* Test process id.         */
  int              state;           /* Current state of test.   */
} instanceInfoT;

static void forceSignal(
  mtiSignalIdT sigid,
  mtiTypeIdT   sigtypeid,
  mtiTypeKindT sigtypekind,
  int          state
)
{
  int           i;
  int           result = 1;
  mtiSignalIdT *elem_list;
  mtiSignalIdT  elem_sigid;
  mtiTypeIdT    elem_typeid;

  switch ( sigtypekind ) {
   case MTI_TYPE_SCALAR:
    switch ( state ) {
      case 0:
       result = mti_ForceSignal(sigid, "42", -1, MTI_FORCE_FREEZE, -1, 1);
       break;
      case 2:
       result = mti_ForceSignal(sigid, "120", 1, MTI_FORCE_FREEZE, 7, -1);
       break;
      case 4:
       result = mti_ForceSignal(sigid, "777", -1, MTI_FORCE_DEPOSIT, -1, 2);
       break;
    }
```

```
 break;
case MTI_TYPE_ARRAY:
 elem_typeid = mti_GetArrayElementType( sigtypeid );
 if ( mti_GetTypeKind( elem_typeid ) == MTI_TYPE_ENUM ) {
     /* NOTE: ASSUMING ARRAY OF LENGTH 4 ! */
   if ( mti_TickLength( elem_typeid ) == 9 ) {  /* ASSUME std_logic */
    switch ( state ) {
     case 0:
      result = mti_ForceSignal( sigid, "ZW1H", -1,
                                MTI_FORCE_FREEZE, -1, -1 );
      break;
     case 2:
      result = mti_ForceSignal( sigid, "LLLL", 1,
                                MTI_FORCE_FREEZE, 7, -1 );
      break;
     case 4:
      result = mti_ForceSignal( sigid, "1-1-", -1,
                                MTI_FORCE_DEPOSIT, -1, 2 );
      break;
    }
   } else {  /* ASSUME bit */
    switch ( state ) {
     case 0:
      result = mti_ForceSignal( sigid, "0011", -1,
                                MTI_FORCE_FREEZE, -1, -1 );
      break;
     case 2:
      result = mti_ForceSignal( sigid, "1000", 1,
                                MTI_FORCE_FREEZE, 7, -1 );
      break;
     case 4:
      result = mti_ForceSignal( sigid, "0010", -1,
                                MTI_FORCE_DEPOSIT, -1, 2 );
      break;
    }
   }
  } else {
   elem_list = mti_GetSignalSubelements( sigid, 0 );
   for ( i = 0; i < mti_TickLength( sigtypeid ); i++ ) {
    elem_sigid  = elem_list[i];
    elem_typeid = mti_GetSignalType( elem_sigid );
    forceSignal( elem_sigid, elem_typeid,
                 mti_GetTypeKind( elem_typeid ), state );
   }
   mti_VsimFree( elem_list );
  }
  break;
 case MTI_TYPE_RECORD:
  elem_list = mti_GetSignalSubelements( sigid, 0 );
  for ( i = 0; i < mti_GetNumRecordElements( sigtypeid ); i++ ) {
   elem_sigid  = elem_list[i];
   elem_typeid = mti_GetSignalType( elem_sigid );
   forceSignal( elem_sigid, elem_typeid,
                mti_GetTypeKind( elem_typeid ), state );
  }
  mti_VsimFree( elem_list );
  break;
 case MTI_TYPE_ENUM:
```

```
        if ( mti_TickLength( sigtypeid ) == 9 ) {    /* ASSUME std_logic */
         switch ( state ) {
          case 0:
           result = mti_ForceSignal( sigid, "'W'", -1,
                                     MTI_FORCE_FREEZE, -1, -1 );
            break;
          case 2:
           result = mti_ForceSignal( sigid, "'0'", 1,
                                     MTI_FORCE_FREEZE, 7, -1 );
            break;
          case 4:
           result = mti_ForceSignal( sigid, "'H'", -1,
                                     MTI_FORCE_DEPOSIT, -1, 2 );
            break;
         }
        } else {
         switch ( state ) {   /* ASSUME bit */
          case 0:
           result = mti_ForceSignal( sigid, "0", -1,
                                     MTI_FORCE_FREEZE, -1, -1 );
            break;
          case 2:
           result = mti_ForceSignal( sigid, "1", 1,
                                     MTI_FORCE_FREEZE, 7, -1 );
            break;
          case 4:
           result = mti_ForceSignal( sigid, "0", -1,
                                      MTI_FORCE_DEPOSIT, -1, 2 );
            break;
         }
        }
       break;
      default:
       break;
     }
     if ( ! result ) {
         fprintf( stderr, "Error in signal force.\n" );
     }
   }
}

static void releaseSignal(
  mtiSignalIdT sigid,
  mtiTypeIdT   sigtypeid,
  mtiTypeKindT sigtypekind
)
{
  int          i;
  mtiSignalIdT *elem_list;
  mtiSignalIdT  elem_sigid;
  mtiTypeIdT    elem_typeid;

  switch ( sigtypekind ) {
    case MTI_TYPE_SCALAR:
    case MTI_TYPE_ENUM:
    case MTI_TYPE_TIME:
      if ( ! mti_ReleaseSignal( sigid ) ) {
        fprintf( stderr, "Error in signal release.\n" );
      }
```

```
        break;
      case MTI_TYPE_ARRAY:
        elem_typeid = mti_GetArrayElementType( sigtypeid );
        if ( mti_GetTypeKind( elem_typeid ) == MTI_TYPE_ENUM ) {
          if ( ! mti_ReleaseSignal( sigid ) ) {
            fprintf( stderr, "Error in signal release.\n" );
          }
        } else {
          elem_list = mti_GetSignalSubelements( sigid, 0 );
          for ( i = 0; i < mti_TickLength( sigtypeid ); i++ ) {
            elem_sigid  = elem_list[i];
            elem_typeid = mti_GetSignalType( elem_sigid );
            releaseSignal( elem_sigid, elem_typeid,
                       mti_GetTypeKind( elem_typeid ) );
          }
          mti_VsimFree( elem_list );
        }
        break;
      case MTI_TYPE_RECORD:
        elem_list = mti_GetSignalSubelements( sigid, 0 );
        for ( i = 0; i < mti_GetNumRecordElements( sigtypeid ); i++ ) {
          elem_sigid  = elem_list[i];
          elem_typeid = mti_GetSignalType( elem_sigid );
          releaseSignal( elem_sigid, elem_typeid,
                     mti_GetTypeKind( elem_typeid ) );
        }
        mti_VsimFree( elem_list );
        break;
      default:
        break;
    }
  }

  static void testForce( void *inst_info )
  {
    instanceInfoT *inst_data = (instanceInfoT *)inst_info;
    signalInfoT   *siginfo;

    switch ( inst_data->state ) {
      case 0:
      case 2:
      case 4:
        for (siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next) {
          forceSignal( siginfo->sigid, siginfo->typeid,
                     siginfo->typekind, inst_data->state );
        }
        break;
      case 1:
      case 3:
      case 5:
        for (siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next) {
          releaseSignal( siginfo->sigid, siginfo->typeid, siginfo->typekind );
        }
        break;
      default:
        break;
    }
```

```
    inst_data->state++;
    mti_ScheduleWakeup( inst_data->proc, 10 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid      = sigid;
  siginfo->name       = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid     = mti_GetSignalType( sigid );
  siginfo->typekind   = mti_GetTypeKind( siginfo->typeid );
  siginfo->direction  = mti_GetSignalMode( sigid );
  siginfo->last_value = mti_GetSignalValueIndirect( sigid, 0 );
  siginfo->child      = 0;
  siginfo->next       = 0;

  /* For records and arrays of composites, we want to set/drive
   * values at the subelement level.  For scalars and arrays of
   * scalars, we want to set/drive values at the top level.
   */
  switch ( siginfo->typekind ) {
    case MTI_TYPE_ARRAY:
      switch( mti_GetTypeKind(mti_GetArrayElementType(siginfo->typeid)) ) {
        case MTI_TYPE_ARRAY:
        case MTI_TYPE_RECORD:
          siginfo->granulate = 1;
          break;
        default:
          siginfo->granulate = 0;
          break;
      }
      break;
    case MTI_TYPE_RECORD:
      siginfo->granulate = 1;
      break;
    default:
      siginfo->granulate = 0;
      break;
  }

  if ( siginfo->granulate ) {
    signalInfoT  * eleminfo;
    signalInfoT  * currinfo;
    int            i;
    mtiSignalIdT * subelem;

    subelem = mti_GetSignalSubelements( siginfo->sigid, 0 );
    for ( i = 0; i < mti_TickLength(siginfo->typeid); i++ ) {
      eleminfo = setupSignal( subelem[i] );
      if ( siginfo->child == 0 ) {
          siginfo->child = eleminfo;
      } else {
        currinfo->next = eleminfo;
      }
      currinfo = eleminfo;
    }
```

```
      mti_VsimFree( subelem );
    }
    return( siginfo );
  }

  static void initInstance( void * param )
  {
    instanceInfoT * inst_data;
    mtiRegionIdT    region;
    mtiSignalIdT    sigid;
    signalInfoT   * curr_info;
    signalInfoT   * siginfo;

    inst_data           = mti_Malloc( sizeof(instanceInfoT) );
    inst_data->sig_info = 0;
    inst_data->state    = 0;
    region              = mti_GetTopRegion();
    for (sigid = mti_FirstSignal( region ); sigid; sigid = mti_NextSignal()) {
      siginfo = setupSignal( sigid );
      if ( inst_data->sig_info == 0 ) {
        inst_data->sig_info = siginfo;
      } else {
        curr_info->next = siginfo;
      }
      curr_info = siginfo;
    }
    inst_data->proc = mti_CreateProcess( "Test Process", testForce,
                                         (void *)inst_data );
    mti_ScheduleWakeup( inst_data->proc, 11 );
  }

  void initForeign(
    mtiRegionIdT       region,   /* The ID of the region in which this     */
                                 /* foreign architecture is instantiated.  */
    char              *param,    /* The last part of the string in the     */
                                 /* foreign attribute.                     */
    mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
  )
  {
    mti_AddLoadDoneCB( initInstance, 0 );
  }
```

HDL code

```
library ieee;
use ieee.std_logic_1164.all;

package typepkg is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )      of integer;

  type rectype is record
    a : bit;
    b : integer;
    c : std_logic;
  end record;

end package typepkg;

entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
  begin
end a;

library ieee;
use ieee.std_logic_1164.all;

use work.typepkg.all;

entity top is
end top;

architecture a of top is

  signal bitsig1      : bit        := '1';
  signal intsig1      : integer    := 21;
  signal stdlogicsig1 : std_logic := 'H';

  signal bitarr1      : bitarray  := "0110";
  signal stdlogicarr1 : std_logic_vector( 1 to 4 ) := "-X0U";
  signal intarr1      : intarray  := ( 10, 11, 12 );

  signal rec1         : rectype   := ( '0', 1, 'X' );

  component for_model
  end component;
  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig1      <= not bitsig1 after 5 ns;
  intsig1      <= intsig1 + 1 after 5 ns;
  stdlogicsig1 <= '-' after 5 ns when stdlogicsig1 = 'H' else
                  'U' after 5 ns when stdlogicsig1 = '-' else
```

```
                        'X' after 5 ns when stdlogicsig1 = 'U' else
                        '0' after 5 ns when stdlogicsig1 = 'X' else
                        '1' after 5 ns when stdlogicsig1 = '0' else
                        'Z' after 5 ns when stdlogicsig1 = '1' else
                        'W' after 5 ns when stdlogicsig1 = 'Z' else
                        'L' after 5 ns when stdlogicsig1 = 'W' else
                        'H' after 5 ns;

    bitarr1         <= not bitarr1 after 5 ns;
    intarr1(1)      <= intarr1(1) + 1 after 5 ns;
    intarr1(2)      <= intarr1(2) + 1 after 5 ns;
    intarr1(3)      <= intarr1(3) + 1 after 5 ns;

    stdlogicarr1(1) <= '-' after 5 ns when stdlogicarr1(1) = 'H' else
                        'U' after 5 ns when stdlogicarr1(1) = '-' else
                        'X' after 5 ns when stdlogicarr1(1) = 'U' else
                        '0' after 5 ns when stdlogicarr1(1) = 'X' else
                        '1' after 5 ns when stdlogicarr1(1) = '0' else
                        'Z' after 5 ns when stdlogicarr1(1) = '1' else
                        'W' after 5 ns when stdlogicarr1(1) = 'Z' else
                        'L' after 5 ns when stdlogicarr1(1) = 'W' else
                        'H' after 5 ns;

    stdlogicarr1(2) <= '-' after 5 ns when stdlogicarr1(2) = 'H' else
                        'U' after 5 ns when stdlogicarr1(2) = '-' else
                        'X' after 5 ns when stdlogicarr1(2) = 'U' else
                        '0' after 5 ns when stdlogicarr1(2) = 'X' else
                        '1' after 5 ns when stdlogicarr1(2) = '0' else
                        'Z' after 5 ns when stdlogicarr1(2) = '1' else
                        'W' after 5 ns when stdlogicarr1(2) = 'Z' else
                        'L' after 5 ns when stdlogicarr1(2) = 'W' else
                        'H' after 5 ns;

    stdlogicarr1(3) <= '-' after 5 ns when stdlogicarr1(3) = 'H' else
                        'U' after 5 ns when stdlogicarr1(3) = '-' else
                        'X' after 5 ns when stdlogicarr1(3) = 'U' else
                        '0' after 5 ns when stdlogicarr1(3) = 'X' else
                        '1' after 5 ns when stdlogicarr1(3) = '0' else
                        'Z' after 5 ns when stdlogicarr1(3) = '1' else
                        'W' after 5 ns when stdlogicarr1(3) = 'Z' else
                        'L' after 5 ns when stdlogicarr1(3) = 'W' else
                        'H' after 5 ns;

    stdlogicarr1(4) <= '-' after 5 ns when stdlogicarr1(4) = 'H' else
                        'U' after 5 ns when stdlogicarr1(4) = '-' else
                        'X' after 5 ns when stdlogicarr1(4) = 'U' else
                        '0' after 5 ns when stdlogicarr1(4) = 'X' else
                        '1' after 5 ns when stdlogicarr1(4) = '0' else
                        'Z' after 5 ns when stdlogicarr1(4) = '1' else
                        'W' after 5 ns when stdlogicarr1(4) = 'Z' else
                        'L' after 5 ns when stdlogicarr1(4) = 'W' else
                        'H' after 5 ns;

    rec1.a <= not rec1.a after 5 ns;
    rec1.b <= rec1.b + 1 after 5 ns;
    rec1.c <= '-' after 5 ns when rec1.c = 'H' else
              'U' after 5 ns when rec1.c = '-' else
              'X' after 5 ns when rec1.c = 'U' else
```

```
                '0' after 5 ns when rec1.c = 'X' else
                '1' after 5 ns when rec1.c = '0' else
                'Z' after 5 ns when rec1.c = '1' else
                'W' after 5 ns when rec1.c = 'Z' else
                'L' after 5 ns when rec1.c = 'W' else
                'H' after 5 ns;

    end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.typepkg
    # Loading work.top(a)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    VSIM 1> add list -w 1 /top/bitsig1
    VSIM 2> add list -w 3 /top/intsig1
    VSIM 3> add list -w 1 /top/stdlogicsig1
    VSIM 4> add list -w 4 /top/bitarr1
    VSIM 5> add list -w 4 /top/stdlogicarr1
    VSIM 6> add list -w 15 /top/intarr1
    VSIM 7> add list -w 10 /top/rec1
    VSIM 8> run 70
    VSIM 9> write list list.out
    VSIM 10> quit -f
    % cat list.out
     ns      /top/bitsig1                       /top/intarr1  /top/rec1
      delta      /top/intsig1
              /top/stdlogicsig1
                    /top/bitarr1
                    /top/stdlogicarr1
      0   +0          1  21 H 0110 -X0U    {10 11 12}   {0 1 X}
      5   +0          0  22 - 1001 U01X    {11 12 13}   {1 2 0}
     10   +0          1  23 U 0110 X1Z0    {12 13 14}   {0 3 1}
     11   +0          0  42 W 0011 ZW1H    {42 42 42}   {0 42 W}
     21   +1          1  43 L 1100 WLZ-    {43 43 43}   {1 43 L}
     26   +0          0  44 H 0011 LHWU    {44 44 44}   {0 44 H}
     31   +0          1  45 - 1100 H-LX    {45 45 45}   {1 45 -}
     32   +0          1 120 0 1000 LLLL    {120 120 120} {1 120 0}
     38   +1          0 121 1 0111 HHHH    {121 121 121} {0 121 1}
     43   +0          1 122 Z 1000 ----    {122 122 122} {1 122 Z}
     48   +0          0 123 W 0111 UUUU    {123 123 123} {0 123 W}
     51   +0          0 777 H 0010 1-1-    {777 777 777} {0 777 H}
     53   +0          0 777 H 0010 1-1-    {777 777 777} {0 777 H}
     56   +0          0 778 - 1101 ZUZU    {778 778 778} {0 778 -}
     57   +0          0 777 H 0010 1-1-    {777 777 777} {0 777 H}
     58   +0          1 777 H 0010 1-1-    {777 777 777} {1 777 H}
     59   +0          0 777 H 0010 1-1-    {777 777 777} {0 777 H}
     61   +1          1 778 - 1101 ZUZU    {778 778 778} {1 778 -}
     66   +0          0 779 U 0010 WXWX    {779 779 779} {0 779 U}
```

mti_RemoveAttrFromVsimTestrecord()

Removes an attribute from a test record of a UCDB database.

Syntax

int = mti_RemoveAttrFromVsimTestrecord(key)

Arguments

Name	Type	Description
key	const char *	Name of the attribute which will be removed.

Return Values

- 0 — upon successful removal of attribute

- -1 — otherwise

Description

Removes an attribute from the vsim implicit test data record.

mti_RemoveEnvCB()

Removes an environment change callback.

Syntax

mti_RemoveEnvCB(func, param)

Arguments

Name	Type	Description
func	mtiEnvCBFuncPtrT	A pointer to a function being called whenever the simulation environment changes
param	void *	The parameter that was specified in the call to mti_AddEnvCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveEnvCB() removes the specified function from the environment change callback list. The param parameter must be the same parameter that was specified in the call to mti_AddEnvCB() when the callback was created.

Examples

FLI code

```
#include
 "mti.h"void envCallback( void * param, void * context )
{  mtiRegionIdT region = (mtiRegionIdT)param;
  mti_PrintFormatted( "Foreign Arch in Region %s: "
                      "the current region is now %s.\n",
                      mti_GetRegionName( region ),
                      mti_GetRegionName( mti_GetCurrentRegion() ) );
  if ( mti_Now() >= 20 ) {
    mti_RemoveEnvCB( envCallback, param );
  }
}
void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this*/
                             /* foreign architecture is instantiated.  */
  char             *param,   /* The last part of the string in the   */
                             /* foreign attribute.                   */
  mtiInterfaceListT *generics, /*A list of generics for the foreign model*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.*/
)
{  mti_AddEnvCB( envCallback, region );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl";
begin
end a;

entity bottom is
end bottom;

architecture b of bottom is
begin
end b;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component bottom is
  end component;

begin

  bot : bottom;

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.bottom(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Foreign Arch in Region i1: the current region is now top.
VSIM 1> env
# sim:/top
VSIM 2> run 10
VSIM 3> env /top/i1
# Foreign Arch in Region i1: the current region is now i1.
# sim:/top/i1
VSIM 4> run 8
VSIM 5> env /top/bot
# Foreign Arch in Region i1: the current region is now bot.
# sim:/top/bot
VSIM 6> run 5
VSIM 7> env /top/i1
# Foreign Arch in Region i1: the current region is now i1.
# sim:/top/i1
VSIM 8> run 2
VSIM 9> env /top
# sim:/top
VSIM 10> quit
```

mti_RemoveLoadDoneCB()

Removes an elaboration done callback.

Syntax

mti_RemoveLoadDoneCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function being called at the end of elaboration
param	void *	The parameter that was specified in the call to mti_AddLoadDoneCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveLoadDoneCB() removes the specified function from the end of elaboration callback list. The param parameter must be the same parameter that was specified in the call to mti_AddLoadDoneCB() when the callback was created.

You must call mti_RemoveLoadDoneCB() from a foreign initialization function in order for the callback removal to take effect. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

Examples

FLI code

```
#include "mti.h"

void loadDoneCallback( void * param )
{
  mtiRegionIdT region = (mtiRegionIdT)param;

  mti_PrintFormatted( "Foreign Arch in Region %s: "
                      "the top-level region is %s.\n",
                      mti_GetRegionName( region ),
                      mti_GetRegionName( mti_GetTopRegion() ) );
}

void initForeign(
  mtiRegionIdT       region,    /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( loadDoneCallback, region );
  if ( ! mti_IsFirstInit() ) {
    mti_RemoveLoadDoneCB( loadDoneCallback, region );
  }
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl";
begin
end a;

entity bottom is
end bottom;

architecture b of bottom is
begin
end b;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

  component bottom is
  end component;

begin

  bot : bottom;

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.bottom(b)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Foreign Arch in Region i1: the top-level region is top.
VSIM 1> run 10
VSIM 2> restart -f
# Loading ./for_model.sl
VSIM 3> run 10
VSIM 4> quit
```

mti_RemoveQuitCB()

Removes a simulator exit callback.

Syntax

mti_RemoveQuitCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function being called at simulator exit
param	void *	The parameter that was specified in the call to mti_AddQuitCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveQuitCB() removes the specified function from the simulator exit callback list. The param parameter must be the same parameter that was specified in the call to mti_AddQuitCB() when the callback was created.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

void quitCallback( void * param )
{
  if ( param ) {
    mti_PrintFormatted( "Cleaning up %s for simulator exit ...\n",
                        (char *)param );
    free( param );
  } else {
    mti_PrintFormatted( "Exiting simulator ...\n" );
  }
}

void loadDoneCallback( void * param )
{
  if ( (int)param == 1 ) {
    mti_RemoveQuitCB( quitCallback, 0 );
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  char * instance_info;

  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddQuitCB( quitCallback, instance_info );
  mti_AddQuitCB( quitCallback, 0 );
  if ( mti_IsFirstInit() ) {
    mti_AddLoadDoneCB( loadDoneCallback, 0 );
  } else {
    mti_AddLoadDoneCB( loadDoneCallback, (void *)1 );
  }
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin
  i1 : for_model;
  s1 <= not s1 after 5 ns;
end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> quit
# Exiting simulator ...
# Cleaning up for_model for simulator exit ...
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> restart -f
# Loading ./for_model.sl
VSIM 3> run 10
VSIM 4> quit
# Cleaning up for_model for simulator exit ...
```

mti_RemoveRestartCB()

Removes a simulator restart callback.

Syntax

mti_RemoveRestartCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function being called at simulator restart
param	void *	The parameter that was specified in the call to mti_AddRestartCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveRestartCB() removes the specified function from the simulator restart callback list. The param parameter must be the same parameter that was specified in the call to mti_AddRestartCB() when the callback was created.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

void restartCallback( void * param )
{
  if ( param ) {
    mti_PrintFormatted( "Cleaning up %s for simulator restart ...\n",
                        (char *)param );
    free( param );
  } else {
    mti_PrintMessage( "Restarting simulator ...\n" );
  }
}

void loadDoneCallback( void * param )
{
  if ( (int)param == 1 ) {
    mti_RemoveRestartCB( restartCallback, 0 );
  }
}

void initForeign(
  mtiRegionIdT       region,    /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  char * instance_info;

  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddRestartCB( restartCallback, instance_info );
  mti_AddRestartCB( restartCallback, 0 );
  if ( mti_IsFirstInit() ) {
    mti_AddLoadDoneCB( loadDoneCallback, 0 );
  } else {
    mti_AddLoadDoneCB( loadDoneCallback, (void *)1 );
  }
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 30
VSIM 2> restart -f
# Restarting simulator ...
# Cleaning up for_model for simulator restart ...
# Loading ./for_model.sl
VSIM 3> run 45
VSIM 4> restart -f
# Cleaning up for_model for simulator restart ...
# Loading ./for_model.sl
VSIM 5> run 10
VSIM 6> quit
```

mti_RemoveRestoreCB()

Removes a simulator restore callback.

Syntax

mti_RemoveRestoreCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function being called at simulator restore
param	void *	The parameter that was specified in the call to mti_AddRestoreCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveRestoreCB() removes the specified function from the simulator restore callback list. The param parameter must be the same parameter that was specified in the call to mti_AddRestoreCB() when the callback was created.

You must call mti_RemoveRestoreCB() from the foreign initialization function in order for the callback to take effect. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  if ( param ) {
    strcpy( inst_info, mti_RestoreString() );
    mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
  } else {
    mti_PrintMessage( "Restore in progress ...\n" );
  }
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,    /* The ID of the region in which this     */
                                /* foreign architecture is instantiated.  */
  char              *param,     /* The last part of the string in the     */
                                /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports      /* A list of ports for the foreign model.   */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, 0 );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
  if ( strcmp( param, "for_model" ) == 0 ) {
    mti_RemoveRestoreCB( restoreCallback, 0 );
  }
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
# Saving instance info "for_model"
VSIM 3> run 50
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 14:55:48 2000
# Restoring state at time 20 ns, iteration 1
# Restored instance info "for_model"
VSIM 5> run 15
VSIM 6> quit
# Cleaning up...
```

mti_RemoveRestoreDoneCB()

Removes a simulator restore done callback.

Syntax

mti_RemoveRestoreDoneCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function being called at simulator restore done
param	void *	The parameter that was specified in the call to mti_AddRestoreDoneCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveRestoreDoneCB() removes the specified function from the simulator restore done callback list. The param parameter must be the same parameter that was specified in the call to mti_AddRestoreDoneCB() when the callback was created.

You must call mti_RemoveRestoreDoneCB() from a foreign initialization function in order for the callback to take effect. You specify a foreign initialization function either in the foreign attribute string of a foreign architecture or in the -foreign string option of a vsim command.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
  mti_SaveString( inst_info );
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", inst_info );
}

void restoreDoneCallback( void * param )
{
  char * inst_info = (char *)param;
  if ( param ) {
    mti_PrintFormatted( "\"%s\": Restore complete\n", inst_info );
  } else {
    mti_PrintMessage( "Restore is done.\n" );
  }
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddRestoreDoneCB( restoreDoneCallback, instance_info );
  mti_AddRestoreDoneCB( restoreDoneCallback, 0 );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
  if ( strcmp( param, "for_model" ) == 0 ) {
    mti_RemoveRestoreDoneCB( restoreDoneCallback, 0 );
```

```
      }
    }
```

HDL code

```
    entity for_model is
    end for_model;

    architecture a of for_model is
      attribute foreign of a : architecture is
        "initForeign for_model.sl; for_model";
    begin
    end a;

    entity top is
    end top;

    architecture a of top is

      signal s1 : bit := '0';

      component for_model is
      end component;

      for all : for_model use entity work.for_model(a);

    begin

      i1 : for_model;

      s1 <= not s1 after 5 ns;

    end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
# Saving instance info "for_model"
VSIM 3> run 45
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 15:15:02 2000
# Restoring state at time 20 ns, iteration 1
# Restored instance info "for_model"
# "for_model": Restore complete
VSIM 5> run 15
VSIM 6> quit
# Cleaning up...
```

mti_RemoveSaveCB()

Removes a simulator checkpoint callback.

Syntax

mti_RemoveSaveCB(func, param)

Arguments

Name	Type	Description
func	mtiVoidFuncPtrT	A pointer to a function being called at simulator checkpoint
param	void *	The parameter that was specified in the call to mti_AddSaveCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveSaveCB() removes the specified function from the simulator checkpoint callback list. The param parameter must be the same parameter that was specified in the call to mti_AddSaveCB() when the callback was created.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char * inst_info = (char *)param;
  if ( param ) {
    mti_PrintFormatted( "Saving instance info \"%s\"\n", inst_info );
    mti_SaveString( inst_info );
  } else {
    mti_PrintFormatted( "Save in progress ...\n" );
  }
}

void restoreCallback( void * param )
{
  char * inst_info = (char *)param;
  strcpy( inst_info, mti_RestoreString() );
  mti_PrintFormatted( "Restored instance info \"%s\"\n", instance_info );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void loadDoneCallback( void * param )
{
  if ( (int)param == 1 ) {
    mti_RemoveSaveCB( saveCallback, 0 );
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instance_info = malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddSaveCB( saveCallback, 0 );
  mti_AddRestoreCB( restoreCallback, instance_info );
  mti_AddQuitCB( cleanupCallback, instance_info );
  mti_AddRestartCB( cleanupCallback, instance_info );
  if ( mti_IsFirstInit() ) {
    mti_AddLoadDoneCB( loadDoneCallback, 0 );
```

```
    } else {
      mti_AddLoadDoneCB( loadDoneCallback, (void *)1 );
    }
  }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 35
VSIM 2> checkpoint cpfile
# Saving instance info "for_model"
# Save in progress ...
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 15:31:29 2000
# Restoring state at time 35 ns, iteration 1
# Restored instance info "for_model"
VSIM 5> run 15
VSIM 6> restart -f
# Cleaning up...
# Loading ./for_model.sl
VSIM 7> run 50
VSIM 8> checkpoint cp2
# Saving instance info "for_model"
VSIM 9> run 35
VSIM 10> restore cp2
# Loading checkpoint/restore data from file "cp2"
# Checkpoint created Fri Jul  7 15:31:48 2000
# Restoring state at time 50 ns, iteration 1
# Restored instance info "for_model"
VSIM 11> run 10
VSIM 12> quit
# Cleaning up...
```

mti_RemoveSimStatusCB()

Removes a simulator run status change callback.

Syntax

mti_RemoveSimStatusCB(func, param)

Arguments

Name	Type	Description
func	mtiSimStatusCBFuncPtrT	A pointer to a function being called at simulator run status change
param	void *	The parameter that was specified in the call to mti_AddSimStatusCB() when the callback was created

Return Values

Nothing

Description

mti_RemoveSimStatusCB() removes the specified function from the simulator run status change callback list. The param parameter must be the same parameter that was specified in the call to mti_AddSimStatusCB() when the callback was created.

Examples

FLI code

```
#include <mti.h>

void simStatusCallback( void * param, int run_status )
{
  mtiRegionIdT region = (mtiRegionIdT)param;

  mti_PrintFormatted( "Time [%ld,%ld]: Region %s: the simulator %s\n",
                       mti_NowUpper(), mti_Now(),
                       mti_GetRegionName( region ),
                       (run_status == 1) ? "is about to run" :
                                           "just completed a run" );
  if ( mti_Now() >= 25 ) {
    mti_RemoveSimStatusCB( simStatusCallback, param );
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddSimStatusCB( simStatusCallback, region );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,0]: Region i1: the simulator is about to run
# Time [0,15]: Region i1: the simulator just completed a run
VSIM 2> run 5
# Time [0,15]: Region i1: the simulator is about to run
# Time [0,20]: Region i1: the simulator just completed a run
VSIM 3> run 8
# Time [0,20]: Region i1: the simulator is about to run
# Time [0,28]: Region i1: the simulator just completed a run
VSIM 4> run 27
VSIM 5> echo $now
# 55
VSIM 6> run 15
VSIM 7> quit
```

mti_RestoreBlock()

Gets a block of data from the checkpoint file.

Syntax

mti_RestoreBlock(ptr)

Arguments

Name	Type	Description
ptr	char *	A pointer to the place where the block of data is to be restored

Return Values

Nothing

Description

mti_RestoreBlock() restores a block of data from the checkpoint file to the address pointed to by the ptr parameter. The size of the data block restored is the same as the size that was saved by the corresponding mti_SaveBlock() call.

You should call this function only from a restore callback function, not from an initialization procedure.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char    tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long    tmp_long  = 123456;
  short   tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char               *param,   /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_RestoreChar()

Gets a byte of data from the checkpoint file.

Syntax

value = mti_RestoreChar()

Arguments

None

Return Values

Name	Type	Description
value	char	A byte of data

Description

mti_RestoreChar() returns a byte of data from the checkpoint file.

You should call this function should be called only from a restore callback function, not from an initialization procedure.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_RestoreLong()

Gets sizeof(long) bytes of data from the checkpoint file.

Syntax

value = mti_RestoreLong()

Arguments

None

Return Values

Name	Type	Description
value	long	Sizeof(long) bytes of data

Description

mti_RestoreLong() returns sizeof(long) bytes of data from the checkpoint file.

You should call This function only from a restore callback function, not from an initialization procedure.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char              *param,   /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_RestoreProcess()

Restores a process that was created by mti_CreateProcess() or
mti_CreateProcessWithPriority().

Syntax

mti_RestoreProcess(process_id, name, func, param)

Arguments

Name	Type	Description
process_id	mtiProcessIdT	A handle to a process created by mti_CreateProcess() or mti_CreateProcessWithPriority()
name	char *	The name of the process as specified to mti_CreateProcess() or mti_CreateProcessWithPriority()
func	mtiVoidFuncPtrT	The callback function as specified to mti_CreateProcess() or mti_CreateProcessWithPriority()
param	void *	The parameter as specified to mti_CreateProcess() or mti_CreateProcessWithPriority()

Return Values

Nothing

Description

mti_RestoreProcess() restores a process that was created by mti_CreateProcess() or
mti_CreateProcessWithPriority(). The first parameter is the handle to the process that was
returned from the original call to mti_CreateProcess() or mti_CreateProcessWithPriority(). The
remaining parameters are the same parameters as in the original call to mti_CreateProcess() or
mti_CreateProcessWithPriority().

You must call mti_RestoreProcess() to restore each process that was created by
mti_CreateProcess() or mti_CreateProcessWithPriority() as the callback function address may
be different after a restore.

You should call this function only from a restore callback function, not from an initialization
procedure.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  mtiProcessIdT procid;
  mtiSignalIdT  sigid;
  mtiDriverIdT  drvid;
} instanceInfoT;

static instanceInfoT * inst_info;

  char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;
  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:           retval = "?";    break;
  }
  return retval;
}

void driveSignal( void * param )
{
  char          * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;

  region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid));
  sigval = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      region_name, mti_GetSignalName( inst->sigid ),
                      convertStdLogicValue( sigval ) );
  mti_VsimFree( region_name );
```

```
  switch ( sigval ) {
    case STD_LOGIC_U:  sigval = STD_LOGIC_X;  break;
    case STD_LOGIC_X:  sigval = STD_LOGIC_0;  break;
    case STD_LOGIC_0:  sigval = STD_LOGIC_1;  break;
    case STD_LOGIC_1:  sigval = STD_LOGIC_Z;  break;
    case STD_LOGIC_Z:  sigval = STD_LOGIC_W;  break;
    case STD_LOGIC_W:  sigval = STD_LOGIC_L;  break;
    case STD_LOGIC_L:  sigval = STD_LOGIC_H;  break;
    case STD_LOGIC_H:  sigval = STD_LOGIC_D;  break;
    case STD_LOGIC_D:  sigval = STD_LOGIC_U;  break;
    default:           sigval = STD_LOGIC_U;  break;
  }
  mti_ScheduleDriver( inst->drvid, sigval, 5, MTI_INERTIAL );
}

void saveCallback( void * param )
{
  mti_SaveBlock( (char *)&inst_info, sizeof(inst_info) );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&inst_info );
  mti_RestoreProcess(inst_info->procid, "sigDriver", driveSignal, inst_info);
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  if ( mti_IsFirstInit() ) {
    inst_info = (instanceInfoT *)mti_Malloc( sizeof(instanceInfoT) );
    inst_info->sigid  = mti_FindSignal( "/top/s1" );
    inst_info->drvid  = mti_CreateDriver( inst_info->sigid );
    inst_info->procid = mti_CreateProcess( "sigDriver",
                                           driveSignal, inst_info );
    mti_Sensitize( inst_info->procid, inst_info->sigid, MTI_EVENT );
    mti_SetDriverOwner( inst_info->drvid, inst_info->procid );
  }
  mti_AddSaveCB( saveCallback, 0 );
  mti_AddRestoreCB( restoreCallback, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal s1 : std_logic := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 30
# Time [0,0] delta 0: Signal /top/s1 is '0'
# Time [0,5] delta 0: Signal /top/s1 is '1'
# Time [0,10] delta 0: Signal /top/s1 is 'Z'
# Time [0,15] delta 0: Signal /top/s1 is 'W'
# Time [0,20] delta 0: Signal /top/s1 is 'L'
# Time [0,25] delta 0: Signal /top/s1 is 'H'
# Time [0,30] delta 0: Signal /top/s1 is '-'
VSIM 2> checkpoint cpfile
VSIM 3> run 20
# Time [0,35] delta 0: Signal /top/s1 is 'U'
# Time [0,40] delta 0: Signal /top/s1 is 'X'
# Time [0,45] delta 0: Signal /top/s1 is '0'
# Time [0,50] delta 0: Signal /top/s1 is '1'
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:48:29 2000
# Restoring state at time 30 ns, iteration 1
VSIM 5> run 25
# Time [0,35] delta 0: Signal /top/s1 is 'U'
# Time [0,40] delta 0: Signal /top/s1 is 'X'
# Time [0,45] delta 0: Signal /top/s1 is '0'
# Time [0,50] delta 0: Signal /top/s1 is '1'
# Time [0,55] delta 0: Signal /top/s1 is 'Z'
VSIM 6> quit
```

mti_RestoreShort()

Gets sizeof(short) bytes of data from the checkpoint file.

Syntax

value = mti_RestoreShort()

Arguments

None

Return Values

Name	Type	Description
value	short	Sizeof(short) bytes of data

Description

mti_RestoreShort() returns sizeof(short) bytes of data from the checkpoint file.

You should call this function only from a restore callback function, not from an initialization procedure.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char    tmp_char  = 'Z';
  char * tmp_str    = "Howdy";
  long    tmp_long  = 123456;
  short   tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_RestoreString()

Gets a null-terminated string from the checkpoint file.

Syntax

value = mti_RestoreString()

Arguments

None

Return Values

Name	Type	Description
value	char *	A null-terminated string

Description

mti_RestoreString() returns a null-terminated string from the checkpoint file. If the size of the string is less than or equal to 1024 bytes (including the NULL), then the string must be copied if it is to be used later because it will be overwritten on the next call to mti_RestoreString(). If the size of the string is greater than 1024 bytes, mti_RestoreString() allocates memory to hold the string. mti_RestoreString() is designed to handle unlimited size strings. The returned pointer must not be freed.

You should call this function only from a restore callback function, not from an initialization procedure.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_SaveBlock()

Saves a block of data to the checkpoint file.

Syntax

mti_SaveBlock(ptr, size)

Arguments

Name	Type	Description
ptr	char *	A pointer to a block of data
size	unsigned long	The size of the data to be saved

Return Values

Nothing

Description

mti_SaveBlock() saves the specified block of data to the checkpoint file.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char    tmp_char  = 'Z';
  char * tmp_str    = "Howdy";
  long    tmp_long  = 123456;
  short   tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this   */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the   */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_SaveChar()

Saves a byte of data to the checkpoint file.

Syntax

mti_SaveChar(data)

Arguments

Name	Type	Description
data	char	The byte of data to be saved

Return Values

Nothing

Description

mti_SaveChar() saves the specified byte of data to the checkpoint file.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_SaveLong()

Saves sizeof(long) bytes of data to the checkpoint file.

Syntax

mti_SaveLong(data)

Arguments

Name	Type	Description
data	long	The data to be saved

Return Values

Nothing

Description

mti_SaveLong() saves the specified sizeof(long) bytes of data to the checkpoint file.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char              *param,   /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_SaveShort()

Saves sizeof(short) bytes of data to the checkpoint file.

Syntax

mti_SaveShort(data)

Arguments

Name	Type	Description
data	short	The data to be saved

Return Values

Nothing

Description

mti_SaveShort() saves the specified sizeof(short) bytes of data to the checkpoint file.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_SaveString()

Saves a null-terminated string to the checkpoint file.

Syntax

mti_SaveString(data)

Arguments

Name	Type	Description
data	char *	A pointer to a null-terminated string

Return Values

Nothing

Description

mti_SaveString() saves the specified null-terminated string to the checkpoint file. mti_SaveString() is designed to handle strings of unlimited size.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

static char * instance_info;

void saveCallback( void * param )
{
  char   tmp_char  = 'Z';
  char * tmp_str   = "Howdy";
  long   tmp_long  = 123456;
  short  tmp_short = 587;

  mti_PrintFormatted( "\nSaving instance info \"%s\"\n", instance_info );
  mti_SaveBlock( (char *)&instance_info, sizeof(instance_info) );
  mti_PrintFormatted( "Saving char %c\n", tmp_char );
  mti_SaveChar( tmp_char );
  mti_PrintFormatted( "Saving long %ld\n", tmp_long );
  mti_SaveLong( tmp_long );
  mti_PrintFormatted( "Saving short %d\n", tmp_short );
  mti_SaveShort( tmp_short );
  mti_PrintFormatted( "Saving string %s\n", tmp_str );
  mti_SaveString( tmp_str );
  mti_PrintFormatted( "\n" );
}

void restoreCallback( void * param )
{
  mti_RestoreBlock( (char *)&instance_info );
  mti_PrintFormatted( "\nRestoring instance info \"%s\"\n", instance_info );
  mti_PrintFormatted( "Restoring char %c\n", mti_RestoreChar() );
  mti_PrintFormatted( "Restoring long %ld\n", mti_RestoreLong() );
  mti_PrintFormatted( "Restoring short %d\n", mti_RestoreShort() );
  mti_PrintFormatted( "Restoring string %s\n", mti_RestoreString() );
  mti_PrintFormatted( "\n" );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this    */
                               /* foreign architecture is instantiated. */
  char              *param,    /* The last part of the string in the    */
                               /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.    */
)
{
  instance_info = mti_Malloc( strlen(param) + 1 );
  strcpy( instance_info, param );
  mti_AddSaveCB( saveCallback, instance_info );
  mti_AddRestoreCB( restoreCallback, instance_info );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is
    "initForeign for_model.sl; my_for_model";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
VSIM 2> checkpoint cpfile
#
# Saving instance info "my_for_model"
# Saving char Z
# Saving long 123456
# Saving short 587
# Saving string Howdy
#
VSIM 3> run 40
VSIM 4> restore cpfile
# Loading checkpoint/restore data from file "cpfile"
# Checkpoint created Fri Jul  7 16:09:02 2000
# Restoring state at time 20 ns, iteration 1
#
# Restoring instance info "my_for_model"
# Restoring char Z
# Restoring long 123456
# Restoring short 587
# Restoring string Howdy
#
VSIM 5> run 10
VSIM 6> quit
```

mti_ScheduleDriver()

Schedules a driver to drive a value onto a VHDL signal.

Syntax

```
mti_ScheduleDriver( driver_id, value, delay, mode )
```

Arguments

Name	Type	Description
driver	mtiDriverIdT	A handle to the driver
value	long/void *	For a signal of scalar type, the value to be driven; for a signal of real, time, or array type, a pointer to the value to be driven
delay	mtiDelayT	The delay to be used in terms of the current simulator resolution limit
mode	mtiDriverModeT	Indicates either inertial or transport delay

Return Values

Nothing

Description

mti_ScheduleDriver() schedules a transaction on the specified driver. If the signal being driven is of an array, real, or time type, then the value type is considered to be "void *" instead of "long".

The specified delay value is multiplied by the current simulator resolution limit. For example, if vsim was invoked with -t 10ns and the delay was specified as 5, then the actual delay would be 50 ns.

The mode parameter can be either MTI_INERTIAL or MTI_TRANSPORT.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

typedef enum {
    STD_LOGIC_U,
    STD_LOGIC_X,
    STD_LOGIC_0,
    STD_LOGIC_1,
    STD_LOGIC_Z,
    STD_LOGIC_W,
    STD_LOGIC_L,
    STD_LOGIC_H,
    STD_LOGIC_D
} StdLogicType;

typedef struct {
    mtiDelayT     delay;
    mtiProcessIdT procid;
    mtiSignalIdT  i1_sigid;
    mtiSignalIdT  i2_sigid;
    mtiSignalIdT  i3_sigid;
    mtiSignalIdT  t1_sigid;
    mtiSignalIdT  t2_sigid;
    mtiSignalIdT  t3_sigid;
    mtiDriverIdT  i1_drvid;
    mtiDriverIdT  i2_drvid;
    mtiDriverIdT  i3_drvid;
    mtiDriverIdT  t1_drvid;
    mtiDriverIdT  t2_drvid;
    mtiDriverIdT  t3_drvid;
    long          i1_last_value;
    long          i2_last_value;
    void *        i3_last_value;
    long          t1_last_value;
    long          t2_last_value;
    void *        t3_last_value;
    mtiInt32T     i3_value_length;
    mtiInt32T     t3_value_length;
} instanceInfoT;

#define NS_EXPONENT -9

mtiDelayT convertToNS( mtiDelayT delay ) {
    int exp = NS_EXPONENT - mti_GetResolutionLimit();

    if (exp < 0) {
        /* Simulator resolution limit is coarser than ns.     */
        /* Cannot represent delay accurately, so truncate it. */
        while (exp++) {
            delay /= 10;
        }
    } else {
        /* Simulator resolution limit is finer than ns. */
        while (exp--) {
```

```
            delay *= 10;
        }
    }
    return delay;
}

static long invertBit( long value )
{
    if ( value == 0 ) {
        return 1;
    } else {
        return 0;
    }
}

static void invertBitArray( char * value, mtiInt32T length )
{
    int i;
    for ( i = 0; i < length; i++ ) {
        if ( value[i] == 0 ) {
            value[i] = 1;
        } else {
            value[i] = 0;
        }
    }
}

static long incrStdLogic( mtiInt32T value )
{
    switch ( value ) {
      case STD_LOGIC_U:  return STD_LOGIC_X;
      case STD_LOGIC_X:  return STD_LOGIC_0;
      case STD_LOGIC_0:  return STD_LOGIC_1;
      case STD_LOGIC_1:  return STD_LOGIC_Z;
      case STD_LOGIC_Z:  return STD_LOGIC_W;
      case STD_LOGIC_W:  return STD_LOGIC_L;
      case STD_LOGIC_L:  return STD_LOGIC_H;
      case STD_LOGIC_H:  return STD_LOGIC_D;
      case STD_LOGIC_D:  return STD_LOGIC_U;
      default:           return STD_LOGIC_U;
    }
}

void driveSignal( void * param )
{
    instanceInfoT * inst = param;

    inst->i1_last_value = invertBit( inst->i1_last_value );
    mti_ScheduleDriver( inst->i1_drvid, inst->i1_last_value,
                        convertToNS(5), MTI_INERTIAL );

    inst->i2_last_value = incrStdLogic( inst->i2_last_value );
    mti_ScheduleDriver( inst->i2_drvid, inst->i2_last_value,
                        convertToNS(5), MTI_INERTIAL );

    invertBitArray( inst->i3_last_value, inst->i3_value_length );
    mti_ScheduleDriver( inst->i3_drvid, (long)(inst->i3_last_value),
                        convertToNS(5), MTI_INERTIAL );
```

```
        inst->t1_last_value = invertBit( inst->t1_last_value );
        mti_ScheduleDriver( inst->t1_drvid, inst->t1_last_value,
                            convertToNS(5), MTI_TRANSPORT );

        inst->t2_last_value = incrStdLogic( inst->t2_last_value );
        mti_ScheduleDriver( inst->t2_drvid, inst->t2_last_value,
                            convertToNS(5), MTI_TRANSPORT );

        invertBitArray( inst->t3_last_value, inst->t3_value_length );
        mti_ScheduleDriver( inst->t3_drvid, (long)(inst->t3_last_value),
                            convertToNS(5), MTI_TRANSPORT );

        mti_ScheduleWakeup( inst->procid, inst->delay );
        inst->delay += convertToNS( 1 );
}

void cleanupCallback( void * param )
{
        mti_PrintMessage( "Cleaning up...\n" );
        free( param );
}

void loadDoneCallback( void * param )
{
        instanceInfoT * inst = param;

        inst->i1_last_value  = mti_GetSignalValue( inst->i1_sigid );
        inst->i2_last_value  = mti_GetSignalValue( inst->i2_sigid );
        inst->i3_last_value  = mti_GetArraySignalValue( inst->i3_sigid, 0 );
        inst->i3_value_length = mti_TickLength( mti_GetSignalType(inst->i3_sigid));

        inst->t1_last_value  = mti_GetSignalValue( inst->t1_sigid );
        inst->t2_last_value  = mti_GetSignalValue( inst->t2_sigid );
        inst->t3_last_value  = mti_GetArraySignalValue( inst->t3_sigid, 0 );
        inst->t3_value_length = mti_TickLength( mti_GetSignalType(inst->t3_sigid));
}

void initForeign(
        mtiRegionIdT      region,   /* The ID of the region in which this      */
                                    /* foreign architecture is instantiated.   */
        char             *param,    /* The last part of the string in the      */
                                    /* foreign attribute.                      */
        mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
        mtiInterfaceListT *ports    /* A list of ports for the foreign model.  */
)
{
        instanceInfoT * inst;

        inst = (instanceInfoT *)malloc( sizeof(instanceInfoT) );

        inst->procid = mti_CreateProcess( "SignalDriver", driveSignal, inst );
        inst->delay = convertToNS( 1 );
        mti_ScheduleWakeup( inst->procid, inst->delay );

        inst->i1_sigid  = mti_FindSignal( "/top/i1" );
        inst->i1_drvid  = mti_CreateDriver( inst->i1_sigid );
        mti_SetDriverOwner( inst->i1_drvid, inst->procid );
```

```
        inst->i2_sigid  = mti_FindSignal( "/top/i2" );
        inst->i2_drvid  = mti_CreateDriver( inst->i2_sigid );
        mti_SetDriverOwner( inst->i2_drvid, inst->procid );

        inst->i3_sigid  = mti_FindSignal( "/top/i3" );
        inst->i3_drvid  = mti_CreateDriver( inst->i3_sigid );
        mti_SetDriverOwner( inst->i3_drvid, inst->procid );

        inst->t1_sigid  = mti_FindSignal( "/top/t1" );
        inst->t1_drvid  = mti_CreateDriver( inst->t1_sigid );
        mti_SetDriverOwner( inst->t1_drvid, inst->procid );

        inst->t2_sigid  = mti_FindSignal( "/top/t2" );
        inst->t2_drvid  = mti_CreateDriver( inst->t2_sigid );
        mti_SetDriverOwner( inst->t2_drvid, inst->procid );

        inst->t3_sigid  = mti_FindSignal( "/top/t3" );
        inst->t3_drvid  = mti_CreateDriver( inst->t3_sigid );
        mti_SetDriverOwner( inst->t3_drvid, inst->procid );

        mti_AddLoadDoneCB( loadDoneCallback, inst );
        mti_AddQuitCB( cleanupCallback, inst );
        mti_AddRestartCB( cleanupCallback, inst );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal i1 : bit := '0';
  signal i2 : std_logic := '0';
  signal i3 : bit_vector( 3 downto 0 ) := "1100";

  signal t1 : bit := '0';
  signal t2 : std_logic := '0';
  signal t3 : bit_vector( 3 downto 0 ) := "1100";

  component for_model
  end component;

begin

  forinst : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> add list /top/i1 /top/t1 /top/i2 /top/t2 /top/i3 /top/t3
VSIM 2> run 35
VSIM 3> write list list.out
VSIM 4> quit
# Cleaning up...
% cat list.out
          ns      /top/i1    /top/i3
           delta      /top/t1        /top/t3
                        /top/i2
                        /top/t2
          0  +0        0 0 0 0 1100 1100
          5  +0        0 1 0 1 1100 0011
          6  +0        0 0 0 Z 1100 1100
          8  +0        0 1 0 W 1100 0011
         11  +0        0 0 0 L 1100 1100
         15  +0        1 1 H H 0011 0011
         20  +0        0 0 - - 1100 1100
         26  +0        1 1 U U 0011 0011
         33  +0        0 0 X X 1100 1100
```

mti_ScheduleDriver64()

Schedules a driver to drive a value onto a VHDL signal with a 64-bit delay.

Syntax

mti_ScheduleDriver64(driver_id, value, delay, mode)

Arguments

Name	Type	Description
driver	mtiDriverIdT	A handle to the driver
value	long/void *	For a signal of scalar type, the value to be driven; for a signal of real, time, or array type, a pointer to the value to be driven
delay	mtiTime64T	The delay to be used in terms of the current simulator resolution limit
mode	mtiDriverModeT	Indicates either inertial or transport delay

Return Values

Nothing

Description

mti_ScheduleDriver64() schedules a transaction on the specified driver using a 64-bit delay. If the signal being driven is of an array, real, or time type, then the value type is considered to be "void *" instead of "long".

The specified delay value is multiplied by the current simulator resolution limit. For example, if vsim was invoked with -t 10ns and the delay was specified as 5, then the actual delay would be 50 ns.

The mode parameter can be either MTI_INERTIAL or MTI_TRANSPORT.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

typedef enum {
    STD_LOGIC_U,
    STD_LOGIC_X,
    STD_LOGIC_0,
    STD_LOGIC_1,
    STD_LOGIC_Z,
    STD_LOGIC_W,
    STD_LOGIC_L,
    STD_LOGIC_H,
    STD_LOGIC_D
} StdLogicType;

typedef struct {
    mtiTime64T    delay;
    mtiProcessIdT procid;
    mtiSignalIdT  i1_sigid;
    mtiSignalIdT  i2_sigid;
    mtiSignalIdT  i3_sigid;
    mtiSignalIdT  t1_sigid;
    mtiSignalIdT  t2_sigid;
    mtiSignalIdT  t3_sigid;
    mtiDriverIdT  i1_drvid;
    mtiDriverIdT  i2_drvid;
    mtiDriverIdT  i3_drvid;
    mtiDriverIdT  t1_drvid;
    mtiDriverIdT  t2_drvid;
    mtiDriverIdT  t3_drvid;
    long          i1_last_value;
    long          i2_last_value;
    void *        i3_last_value;
    long          t1_last_value;
    long          t2_last_value;
    void *        t3_last_value;
    mtiInt32T     i3_value_length;
    mtiInt32T     t3_value_length;
} instanceInfoT;

static long invertBit( long value )
{
    if ( value == 0 ) {
        return 1;
    } else {
        return 0;
    }
}

static void invertBitArray( char * value, mtiInt32T length )
{
    int i;
    for ( i = 0; i < length; i++ ) {
        if ( value[i] == 0 ) {
```

```
                value[i] = 1;
            } else {
                value[i] = 0;
            }
        }
    }

    static long incrStdLogic( mtiInt32T value )
    {
        switch ( value ) {
          case STD_LOGIC_U:  return STD_LOGIC_X;
          case STD_LOGIC_X:  return STD_LOGIC_0;
          case STD_LOGIC_0:  return STD_LOGIC_1;
          case STD_LOGIC_1:  return STD_LOGIC_Z;
          case STD_LOGIC_Z:  return STD_LOGIC_W;
          case STD_LOGIC_W:  return STD_LOGIC_L;
          case STD_LOGIC_L:  return STD_LOGIC_H;
          case STD_LOGIC_H:  return STD_LOGIC_D;
          case STD_LOGIC_D:  return STD_LOGIC_U;
          default:           return STD_LOGIC_U;
        }
    }

    void driveSignal( void * param )
    {
        instanceInfoT * inst = param;
        mtiTime64T at_time;

        MTI_TIME64_ASGN( at_time, 1, 2 );

        inst->i1_last_value = invertBit( inst->i1_last_value );
        mti_ScheduleDriver64( inst->i1_drvid, inst->i1_last_value,
                              at_time, MTI_INERTIAL );

        inst->i2_last_value = incrStdLogic( inst->i2_last_value );
        mti_ScheduleDriver64( inst->i2_drvid, inst->i2_last_value,
                              at_time, MTI_INERTIAL );

        invertBitArray( inst->i3_last_value, inst->i3_value_length );
        mti_ScheduleDriver64( inst->i3_drvid, (long)(inst->i3_last_value),
                              at_time, MTI_INERTIAL );

        inst->t1_last_value = invertBit( inst->t1_last_value );
        mti_ScheduleDriver64( inst->t1_drvid, inst->t1_last_value,
                              at_time, MTI_TRANSPORT );

        inst->t2_last_value = incrStdLogic( inst->t2_last_value );
        mti_ScheduleDriver64( inst->t2_drvid, inst->t2_last_value,
                              at_time, MTI_TRANSPORT );

        invertBitArray( inst->t3_last_value, inst->t3_value_length );
        mti_ScheduleDriver64( inst->t3_drvid, (long)(inst->t3_last_value),
                              at_time, MTI_TRANSPORT );

        mti_ScheduleWakeup64( inst->procid, inst->delay );
    }

    void loadDoneCallback( void * param )
```

```
    {
        instanceInfoT * inst = param;

        inst->i1_last_value   = mti_GetSignalValue( inst->i1_sigid );
        inst->i2_last_value   = mti_GetSignalValue( inst->i2_sigid );
        inst->i3_last_value   = mti_GetArraySignalValue( inst->i3_sigid, 0 );
        inst->i3_value_length = mti_TickLength( mti_GetSignalType(inst->i3_sigid));

        inst->t1_last_value   = mti_GetSignalValue( inst->t1_sigid );
        inst->t2_last_value   = mti_GetSignalValue( inst->t2_sigid );
        inst->t3_last_value   = mti_GetArraySignalValue( inst->t3_sigid, 0 );
        inst->t3_value_length = mti_TickLength( mti_GetSignalType(inst->t3_sigid));
    }


    void initForeign(
        mtiRegionIdT        region,   /* The ID of the region in which this    */
                                      /* foreign architecture is instantiated. */
        char               *param,    /* The last part of the string in the    */
                                      /* foreign attribute.                    */
        mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
        mtiInterfaceListT *ports     /* A list of ports for the foreign model. */
    )
    {
        instanceInfoT * inst;

        inst = (instanceInfoT *)mti_Malloc( sizeof(instanceInfoT) );

        inst->procid = mti_CreateProcess( "SignalDriver", driveSignal, inst );
        MTI_TIME64_ASGN( inst->delay, 1, 30 );
        mti_ScheduleWakeup64( inst->procid, inst->delay );

        inst->i1_sigid  = mti_FindSignal( "/top/i1" );
        inst->i1_drvid  = mti_CreateDriver( inst->i1_sigid );
        mti_SetDriverOwner( inst->i1_drvid, inst->procid );

        inst->i2_sigid  = mti_FindSignal( "/top/i2" );
        inst->i2_drvid  = mti_CreateDriver( inst->i2_sigid );
        mti_SetDriverOwner( inst->i2_drvid, inst->procid );

        inst->i3_sigid  = mti_FindSignal( "/top/i3" );
        inst->i3_drvid  = mti_CreateDriver( inst->i3_sigid );
        mti_SetDriverOwner( inst->i3_drvid, inst->procid );

        inst->t1_sigid  = mti_FindSignal( "/top/t1" );
        inst->t1_drvid  = mti_CreateDriver( inst->t1_sigid );
        mti_SetDriverOwner( inst->t1_drvid, inst->procid );

        inst->t2_sigid  = mti_FindSignal( "/top/t2" );
        inst->t2_drvid  = mti_CreateDriver( inst->t2_sigid );
        mti_SetDriverOwner( inst->t2_drvid, inst->procid );

        inst->t3_sigid  = mti_FindSignal( "/top/t3" );
        inst->t3_drvid  = mti_CreateDriver( inst->t3_sigid );
        mti_SetDriverOwner( inst->t3_drvid, inst->procid );

        mti_AddLoadDoneCB( loadDoneCallback, inst );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is
end top;

architecture a of top is

  signal i1 : bit := '0';
  signal i2 : std_logic := '0';
  signal i3 : bit_vector( 3 downto 0 ) := "1100";

  signal t1 : bit := '0';
  signal t2 : std_logic := '0';
  signal t3 : bit_vector( 3 downto 0 ) := "1100";

  component for_model
  end component;

begin

  forinst : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.6

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> add list /top/i1 /top/t1 /top/i2 /top/t2 /top/i3 /top/t3
VSIM 2> run 30 sec
VSIM 3> write list list.out
VSIM 4> quit
% cat list.out
         ns        /top/i1    /top/i3
           delta      /top/t1       /top/t3
                      /top/i2
                      /top/t2
          0  +0         0 0 0 0 1100 1100
  4294967298  +0         1 1 1 1 0011 0011
  8589934624  +0         0 0 Z Z 1100 1100
  1288490195  +0         1 1 W W 0011 0011
  1717986927  +0         0 0 L L 1100 1100
  2147483660  +0         1 1 H H 0011 0011
  2576980392  +0         0 0 - - 1100 1100
```

mti_ScheduleWakeup()

Schedules a VHDL process to wake up at a specific time.

Syntax

mti_ScheduleWakeup(process_id, delay)

Arguments

Name	Type	Description
process_id	mtiProcessIdT	A handle to a VHDL process
delay	mtiDelayT	The delay to be used in terms of the current simulator resolution limit

Return Values

Nothing

Description

mti_ScheduleWakeup() schedules the specified process to be called after the specified delay. A process can have no more than one pending wake-up call. A call to mti_ScheduleWakeup() cancels a prior pending wake-up call for the specified process regardless of the delay values.

The specified delay value is multiplied by the current simulator resolution limit. For example, if vsim was invoked with -t 10ns and the delay was specified as 5, then the actual delay would be 50 ns.

The process_id must be a handle to a process that was created by mti_CreateProcess() or mti_CreateProcessWithPriority().

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef struct {
  mtiDelayT     delay;
  mtiProcessIdT procid;
  mtiSignalIdT  i1_sigid;
  mtiSignalIdT  t1_sigid;
  mtiDriverIdT  i1_drvid;
  mtiDriverIdT  t1_drvid;
  long          i1_last_value;
  long          t1_last_value;
} instanceInfoT;

static long invertBit( long value )
{
  if ( value == 0 ) {
    return 1;
  } else {
    return 0;
  }
}

void driveSignal( void * param )
{
  instanceInfoT * inst = ( instanceInfoT * ) param;
  inst->i1_last_value = invertBit( inst->i1_last_value );
  mti_ScheduleDriver( inst->i1_drvid, inst->i1_last_value, 5, MTI_INERTIAL );

  inst->t1_last_value = invertBit( inst->t1_last_value );
  mti_ScheduleDriver( inst->t1_drvid, inst->t1_last_value, 5, MTI_TRANSPORT );

  mti_ScheduleWakeup( inst->procid, inst->delay );
  inst->delay++;
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void loadDoneCallback( void * param )
{
  instanceInfoT * inst = ( instanceInfoT * ) param;
  inst->i1_last_value   = mti_GetSignalValue( inst->i1_sigid );
  inst->t1_last_value   = mti_GetSignalValue( inst->t1_sigid );
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char             *param,     /* The last part of the string in the     */
                               /* foreign attribute.                     */
```

```
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instanceInfoT * inst;

  inst = (instanceInfoT *)malloc( sizeof(instanceInfoT) );

  inst->procid = mti_CreateProcess( "SignalDriver", driveSignal, inst );
  inst->delay = 1;
  mti_ScheduleWakeup( inst->procid, inst->delay );

  inst->i1_sigid  = mti_FindSignal( "/top/i1" );
  inst->i1_drvid  = mti_CreateDriver( inst->i1_sigid );
  mti_SetDriverOwner( inst->i1_drvid, inst->procid );

  inst->t1_sigid  = mti_FindSignal( "/top/t1" );
  inst->t1_drvid  = mti_CreateDriver( inst->t1_sigid );
  mti_SetDriverOwner( inst->t1_drvid, inst->procid );

  mti_AddLoadDoneCB( loadDoneCallback, inst );
  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
entity for_model is
end for_model;
architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

entity top is
end top;

architecture a of top is

  signal i1 : bit := '0';
  signal t1 : bit := '0';

  component for_model
  end component;

begin

  forinst : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.6

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> add list /top/i1 /top/t1
VSIM 2> run 35
VSIM 3> write list list.out
VSIM 4> quit
# Cleaning up...
% cat list.out
        ns      /top/i1
          delta      /top/t1
         0  +0          0 0
         5  +0          0 1
         6  +0          0 0
         8  +0          0 1
        11  +0          0 0
        15  +0          1 1
        20  +0          0 0
        26  +0          1 1
        33  +0          0 0
```

mti_ScheduleWakeup64()

Schedules a VHDL process to wake up at a specific time using a 64-bit delay.

Syntax

mti_ScheduleWakeup64(process_id, delay)

Arguments

Name	Type	Description
process_id	mtiProcessIdT	A handle to a VHDL process
delay	mtiTime64T	The delay to be used in terms of the current simulator resolution limit

Return Values

Nothing

Description

mti_ScheduleWakeup64() schedules the specified process to be called after the specified 64-bit delay. A process can have no more than one pending wake-up call. A call to mti_ScheduleWakeup64() cancels a prior pending wake-up call for the specified process regardless of the delay values.

The specified delay value is multiplied by the current simulator resolution limit. For example, if vsim was invoked with -t 10ns and the delay was specified as 5, then the actual delay would be 50 ns.

The process_id must be a handle to a process that was created by mti_CreateProcess() or mti_CreateProcessWithPriority().

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef struct {
    mtiTime64T    delay;
    mtiProcessIdT procid;
    mtiSignalIdT  i1_sigid;
    mtiSignalIdT  t1_sigid;
    mtiDriverIdT  i1_drvid;
    mtiDriverIdT  t1_drvid;
    long          i1_last_value;
    long          t1_last_value;
} instanceInfoT;

static long invertBit( long value )
{
    if ( value == 0 ) {
        return 1;
    } else {
        return 0;
    }
}

void driveSignal( void * param )
{
    instanceInfoT * inst = ( instanceInfoT * ) param;
    mtiTime64T curr_time;

    mti_PrintFormatted( "Time %s: Executing driveSignal()\n",
                        mti_Image( mti_NowIndirect( &curr_time ),
                                   mti_CreateTimeType() ) );

    inst->i1_last_value = invertBit( inst->i1_last_value );
    mti_ScheduleDriver( inst->i1_drvid, inst->i1_last_value, 5, MTI_INERTIAL );

    inst->t1_last_value = invertBit( inst->t1_last_value );
    mti_ScheduleDriver( inst->t1_drvid, inst->t1_last_value, 5, MTI_TRANSPORT );

    mti_ScheduleWakeup64( inst->procid, inst->delay );
    MTI_TIME64_ASGN( inst->delay,
                     MTI_TIME64_HI32(inst->delay),
                     MTI_TIME64_LO32(inst->delay) + 1 );
}

void cleanupCallback( void * param )
{
    mti_PrintMessage( "Cleaning up...\n" );
    free( param );
}

void loadDoneCallback( void * param )
{
    instanceInfoT * inst = ( instanceInfoT * ) param;
    inst->i1_last_value = mti_GetSignalValue( inst->i1_sigid );
```

```
        inst->t1_last_value = mti_GetSignalValue( inst->t1_sigid );
    }

    void initForeign(
        mtiRegionIdT        region,   /* The ID of the region in which this     */
                                      /* foreign architecture is instantiated.  */
        char               *param,    /* The last part of the string in the     */
                                      /* foreign attribute.                     */
        mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
        mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
    )
    {
        instanceInfoT * inst;

        inst = (instanceInfoT *)malloc( sizeof(instanceInfoT) );

        inst->procid = mti_CreateProcess( "SignalDriver", driveSignal, inst );
        MTI_TIME64_ASGN( inst->delay, 1, 1 );
        mti_ScheduleWakeup64( inst->procid, inst->delay );

        inst->i1_sigid = mti_FindSignal( "/top/i1" );
        inst->i1_drvid = mti_CreateDriver( inst->i1_sigid );
        mti_SetDriverOwner( inst->i1_drvid, inst->procid );

        inst->t1_sigid = mti_FindSignal( "/top/t1" );
        inst->t1_drvid = mti_CreateDriver( inst->t1_sigid );
        mti_SetDriverOwner( inst->t1_drvid, inst->procid );

        mti_AddLoadDoneCB( loadDoneCallback, inst );
        mti_AddQuitCB( cleanupCallback, inst );
        mti_AddRestartCB( cleanupCallback, inst );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

entity top is
end top;

architecture a of top is

  signal i1 : bit := '0';
  signal t1 : bit := '0';

  component for_model
  end component;

begin

  forinst : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> add list /top/i1 /top/t1
VSIM 2> run 10 sec
# Time {0 ns}: Executing driveSignal()
# Time {4294967297 ns}: Executing driveSignal()
# Time {8589934595 ns}: Executing driveSignal()
VSIM 3> write list list.out
VSIM 4> quit
# Cleaning up...
% cat list.out
        ns       /top/i1
          delta     /top/t1
          0  +0         0 0
          5  +0         1 1
 4294967302  +0         0 0
 8589934600  +0         1 1
```

mti_Sensitize()

Sensitizes a VHDL process to a VHDL or SystemC signal.

Syntax

mti_Sensitize(process_id, signal_id, trigger)

Arguments

Name	Type	Description
process_id	mtiProcessIdT	A handle to a VHDL process
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal
trigger	mtiProcessTriggerT	Indicates either event-based or activity-based triggering

Return Values

Nothing

Description

mti_Sensitize() causes the specified process to be called when the specified signal is updated. If the trigger parameter is MTI_EVENT, then the process is called when the signal changes value. If the trigger parameter is MTI_ACTIVE, then the process is called whenever the signal is active. The tool supports only the MTI_EVENT trigger parameter for SystemC signals.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef struct {
  mtiSignalIdT sigid1;
  mtiSignalIdT sigid2;
} instanceInfoT;

void monitorSignal1( void * param )
{
  instanceInfoT * inst = ( instanceInfoT * ) param;
  mti_PrintFormatted( "Time [%d,%d]:", mti_NowUpper(), mti_Now() );
  mti_PrintFormatted( "  %s = %s\n", mti_GetSignalName( inst->sigid1 ),
                    mti_SignalImage( inst->sigid1 ) );
}

void monitorSignal2( void * param )
{
  instanceInfoT * inst = ( instanceInfoT * ) param;
  mti_PrintFormatted( "Time [%d,%d]:", mti_NowUpper(), mti_Now() );
  mti_PrintFormatted( "  %s = %s\n", mti_GetSignalName( inst->sigid2 ),
                    mti_SignalImage( inst->sigid2 ) );
}

void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this      */
                              /* foreign architecture is instantiated.   */
  char             *param,    /* The last part of the string in the      */
                              /* foreign attribute.                      */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  instanceInfoT * inst;
  mtiProcessIdT   procid;

  inst        = (instanceInfoT *)malloc( sizeof(instanceInfoT) );

  inst->sigid1 = mti_FindSignal( "/top/s1" );
  procid       = mti_CreateProcess( "s1Monitor", monitorSignal1, inst );
  mti_Sensitize( procid, inst->sigid1, MTI_EVENT );

  inst->sigid2 = mti_FindSignal( "/top/s2" );
  procid       = mti_CreateProcess( "s2Monitor", monitorSignal2, inst );
  mti_Sensitize( procid, inst->sigid2, MTI_ACTIVE );

  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst );
```

```
    }
```

HDL code

```
    entity for_model is
    end for_model;
    architecture a of for_model is
      attribute foreign of a : architecture is "initForeign for_model.sl;";
    begin
    end a;

    entity top is
    end top;

    architecture a of top is

      signal s1 : bit := '0';
      signal s2 : bit := '0';

      component for_model
      end component;

    begin

      s1 <= not s1 after 5 ns;

      forinst : for_model;

      p1 : process
        begin
          wait for 2 ns;
          s2 <= '0';
          wait for 5 ns;
          s2 <= '1';
          wait for 3 ns;
          s2 <= '1';
          wait for 4 ns;
          s2 <= '0';
        end process;
    end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 20
# Time [0,0]:  s2 = '0'
# Time [0,0]:  s1 = '0'
# Time [0,2]:  s2 = '0'
# Time [0,5]:  s1 = '1'
# Time [0,7]:  s2 = '1'
# Time [0,10]:  s1 = '0'
# Time [0,10]:  s2 = '1'
# Time [0,14]:  s2 = '0'
# Time [0,15]:  s1 = '1'
# Time [0,16]:  s2 = '0'
# Time [0,20]:  s1 = '0'
VSIM 2> quit
# Cleaning up...
```

mti_SetDriverOwner()

Sets the owning process of a driver.

Syntax

mti_SetDriverOwner(driver_id, process_id)

Arguments

Name	Type	Description
driver_id	mtiDriverIdT	A handle to a VHDL driver
process_id	mtiProcessIdT	A handle to a VHDL process

Return Values

Nothing

Description

mti_SetDriverOwner() makes the specified process the owner of the specified driver.

Normally, mti_CreateDriver() makes the <MTI_foreign_architecture> process the owner of a new driver. When using mti_CreateDriver() it is necessary to follow up with a call to mti_SetDriverOwner(); otherwise, the "drivers" command and the Dataflow window may give unexpected or incorrect information regarding FLI-created drivers.

Examples

FLI code

```c
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} mySigType;

char *std_logic_lits[9] =
{ "'U'", "'X'", "'0'", "'1'", "'Z'", "'W'", "'L'", "'H'", "'-'" };

typedef struct {
  mtiSignalIdT sigid1;
  mtiSignalIdT sigid2;
  mtiDriverIdT drvid1;
  mtiDriverIdT drvid2;
} instanceInfoT;

/* This function inverts mySig1 every 5 ns. */
void driveSignal1( void * param )
{
  char          * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;

  sigval = mti_GetSignalValue( inst->sigid1 );

  switch ( sigval ) {
   case STD_LOGIC_U:
    mti_ScheduleDriver( inst->drvid1, STD_LOGIC_0, 0, MTI_INERTIAL );
    break;
   case STD_LOGIC_0:
    mti_ScheduleDriver( inst->drvid1, STD_LOGIC_1, 5, MTI_INERTIAL );
    break;
   case STD_LOGIC_1:
    mti_ScheduleDriver( inst->drvid1, STD_LOGIC_0, 5, MTI_INERTIAL );
    break;
   case STD_LOGIC_X:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
    mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is UNKNOWN\n",
                    mti_NowUpper(), mti_Now(), mti_Delta(),
                    region_name, mti_GetSignalName( inst->sigid1 ) );
    mti_VsimFree( region_name );
    break;
   default:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid1));
    mti_PrintFormatted( "Time [%d,%d] delta %d: "
```

```
                         "Unexpected value %d on signal %s/%s\n",
                         mti_NowUpper(), mti_Now(), mti_Delta(),
                         sigval, region_name,
                         mti_GetSignalName( inst->sigid1 ) );
      mti_VsimFree( region_name );
      break;
    }
  }
}

/* This function inverts mySig2 every 10 ns. */
void driveSignal2( void * param )
{
  char         * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T      sigval;

  sigval = mti_GetSignalValue( inst->sigid2 );

  switch ( sigval ) {
   case STD_LOGIC_U:
    mti_ScheduleDriver( inst->drvid2, STD_LOGIC_0, 0, MTI_INERTIAL );
    break;
   case STD_LOGIC_0:
    mti_ScheduleDriver( inst->drvid2, STD_LOGIC_1, 10, MTI_INERTIAL );
    break;
   case STD_LOGIC_1:
    mti_ScheduleDriver( inst->drvid2, STD_LOGIC_0, 10, MTI_INERTIAL );
    break;
   case STD_LOGIC_X:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
    mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is UNKNOWN\n",
                        mti_NowUpper(), mti_Now(), mti_Delta(),
                        region_name, mti_GetSignalName( inst->sigid2 ) );
    mti_VsimFree( region_name );
    break;
   default:
    region_name = mti_GetRegionFullName(mti_GetSignalRegion(inst->sigid2));
    mti_PrintFormatted( "Time [%d,%d] delta %d: "
                         "Unexpected value %d on signal %s/%s\n",
                         mti_NowUpper(), mti_Now(), mti_Delta(),
                         sigval, region_name,
                         mti_GetSignalName( inst->sigid2 ) );
    mti_VsimFree( region_name );
    break;
  }
}
void cleanupCallback( void * param )
{
  mti_PrintMessage( "Cleaning up...\n" );
  free( param );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
```

```
    mtiInterfaceListT *ports    /* A list of ports for the foreign model.   */
)
{
  instanceInfoT * inst;
  mtiProcessIdT   procid;
  mtiTypeIdT      enum_type;

  inst        = (instanceInfoT *)malloc( sizeof(instanceInfoT) );
  enum_type   = mti_CreateEnumType( 1, 9, std_logic_lits );

  inst->sigid1 = mti_CreateSignal( "mySig1", region, enum_type );
  inst->drvid1 = mti_CreateDriver( inst->sigid1 );
  procid       = mti_CreateProcess( "mySig1Driver", driveSignal1, inst );
  mti_Sensitize( procid, inst->sigid1, MTI_EVENT );
  mti_SetDriverOwner( inst->drvid1, procid );

  inst->sigid2 = mti_CreateSignal( "mySig2", region, enum_type );
  inst->drvid2 = mti_CreateDriver( inst->sigid2 );
  procid       = mti_CreateProcess( "mySig2Driver", driveSignal2, inst );
  mti_Sensitize( procid, inst->sigid2, MTI_EVENT );
  /* Not setting driver owner for driver 2. */

  mti_AddQuitCB( cleanupCallback, inst );
  mti_AddRestartCB( cleanupCallback, inst );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model
  end component;

begin

  s1 <= not s1 after 5 ns;

  forinst : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 5
VSIM 2> drivers /top/forinst/mySig1
# Drivers for /top/forinst/mysig1:
#  1 : Signal /top/forinst/mysig1
#    1 : Driver /top/forinst/mySig1Driver
#         0 at 10 ns
#
VSIM 3> drivers /top/forinst/mySig2
# Drivers for /top/forinst/mysig2:
#  0 : Signal /top/forinst/mysig2
#    0 : Driver /top/forinst/<MTI_foreign_architecture>
#         1 at 10 ns
#
VSIM 4> quit
# Cleaning up...
```

mti_SetSignalValue()

Sets the value of a VHDL signal.

Syntax

mti_SetSignalValue(signal_id, value)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL signal
value	long/void *	For a signal of scalar type, the value to be set; for a signal of real, time, or array type, a pointer to the value to be set

Return Values

Nothing

Description

mti_SetSignalValue() sets the specified VHDL signal to the specified value immediately. The signal can be either an unresolved signal or a resolved signal. Setting the signal marks it as active in the current delta. If the new value is different than the old value, then an event occurs on the signal in the current delta. If the specified signal is of type array, real, or time, then the value type is considered to be "void *" instead of "long".

You cannot use mti_SetSignalValue() to set the value of a signal of type record, but you can use it to set the values on the individual scalar or array subelements.

Setting a resolved signal is not the same as driving it. After a resolved signal is set it may be changed to a new value the next time its resolution function is executed. mti_ScheduleDriver() and mti_ScheduleDriver64() can be used to drive a value onto a signal.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                    * name;
  mtiSignalIdT              sigid;
  mtiTypeIdT                typeid;
} signalInfoT;

typedef struct {
  signalInfoT   * sig_info;     /* List of signals. */
  mtiProcessIdT   proc;         /* Test process id.*/
} instanceInfoT;

static void setValue( mtiSignalIdT sigid, mtiTypeIdT sigtype )
{
  switch ( mti_GetTypeKind( sigtype ) ) {
    case MTI_TYPE_ENUM:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetSignalValue( sigid );
        scalar_val++;
        if (( scalar_val < mti_TickLow( sigtype )) ||
          ( scalar_val > mti_TickHigh( sigtype ))) {
          scalar_val = mti_TickLeft( sigtype );
        }
        mti_SetSignalValue( sigid, (long)scalar_val );
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetSignalValue( sigid );
        scalar_val++;
        mti_SetSignalValue( sigid, (long)scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int             i;
        mtiTypeIdT      elem_type;
        mtiSignalIdT * elem_list;
        elem_type = mti_GetArrayElementType( sigtype );
        switch ( mti_GetTypeKind( elem_type ) ) {
          case MTI_TYPE_SCALAR:
          case MTI_TYPE_PHYSICAL:
            {
              mtiInt32T * array_val = mti_GetArraySignalValue( sigid, 0 );
              for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
                array_val[i]++;
              }
              mti_SetSignalValue( sigid, (long)array_val );
              mti_VsimFree( array_val );
```

```
            }
          break;
        case MTI_TYPE_ARRAY:
        case MTI_TYPE_RECORD:
        default:
          elem_list = mti_GetSignalSubelements( sigid, 0 );
          for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
            setValue( elem_list[i], mti_GetSignalType( elem_list[i] ));
          }
          mti_VsimFree( elem_list );
          break;
        case MTI_TYPE_ENUM:
          if ( mti_TickLength( elem_type ) <= 256 ) {
            char * array_val = mti_GetArraySignalValue( sigid, 0 );
            for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
              array_val[i]++;
              if (( array_val[i] < mti_TickLow( elem_type )) ||
                ( array_val[i] > mti_TickHigh( elem_type ))) {
                array_val[i] = mti_TickLeft( elem_type );
              }
            }
            mti_SetSignalValue( sigid, (long)array_val );
            mti_VsimFree( array_val );
          } else {
            mtiInt32T * array_val = mti_GetArraySignalValue( sigid, 0 );
            for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
              array_val[i]++;
              if (( array_val[i] < mti_TickLow( elem_type )) ||
                ( array_val[i] > mti_TickHigh( elem_type ))) {
                array_val[i] = mti_TickLeft( elem_type );
              }
            }
            mti_SetSignalValue( sigid, (long)array_val );
            mti_VsimFree( array_val );
          }
          break;
        case MTI_TYPE_REAL:
          {
            double * array_val = mti_GetArraySignalValue( sigid, 0 );
            for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
              array_val[i] = array_val[i] + 1.1;
            }
            mti_SetSignalValue( sigid, (long)array_val );
            mti_VsimFree( array_val );
          }
          break;
        case MTI_TYPE_TIME:
          {
            mtiTime64T * array_val = mti_GetArraySignalValue(sigid, 0);
            for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
              MTI_TIME64_ASGN( array_val[i],
                          MTI_TIME64_HI32(array_val[i]),
                          MTI_TIME64_LO32(array_val[i]) + 1 );
            }
            mti_SetSignalValue( sigid, (long)array_val );
            mti_VsimFree( array_val );
          }
          break;
```

```
              }
          }
        break;
      case MTI_TYPE_RECORD:
          {
            int             i;
            mtiSignalIdT * elem_list;
            elem_list = mti_GetSignalSubelements( sigid, 0 );
            for ( i = 0; i < mti_TickLength( sigtype ); i++ ) {
              setValue( elem_list[i], mti_GetSignalType( elem_list[i] ));
            }
            mti_VsimFree( elem_list );
          }
        break;
      case MTI_TYPE_REAL:
        {
          double real_val;
          mti_GetSignalValueIndirect( sigid, &real_val );
          real_val += 1.1;
          mti_SetSignalValue( sigid, (long)(&real_val) );
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T time_val;
          mti_GetSignalValueIndirect( sigid, &time_val );
          MTI_TIME64_ASGN( time_val, MTI_TIME64_HI32(time_val),
                          MTI_TIME64_LO32(time_val) + 1 );
          mti_SetSignalValue( sigid, (long)(&time_val) );
        }
        break;
      default:
        break;
    }
}

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  signalInfoT   *siginfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
    mti_PrintFormatted( "  Signal %s = %s\n", siginfo->name,
                        mti_SignalImage( siginfo->sigid ));
    setValue( siginfo->sigid, siginfo->typeid );
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid  = sigid;
```

```
  siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid = mti_GetSignalType( sigid );
  siginfo->next   = 0;

  return( siginfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data          = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    siginfo = setupSignal( sigid );
    if ( inst_data->sig_info == 0 ) {
      inst_data->sig_info = siginfo;
    }
    else {
      curr_info->next = siginfo;
    }
    curr_info = siginfo;
  }

  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 5 );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

    signal bitsig      : bit        := '1';
    signal intsig      : integer    := 21;
    signal realsig     : real       := 16.35;
    signal timesig     : time       := 5 ns;
    signal stdlogicsig : std_logic  := 'H';

    signal bitarr      : bitarray   := "0110";
    signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    signal intarr      : intarray   := ( 10, 11, 12 );
    signal realarr     : realarray  := ( 11.6, 101.22 );
    signal timearr     : timearray  := ( 15 ns, 6 ns );

    signal rec         : rectype    := ( '0', 1, 3.7, 'H', "1001" );

begin

  inst1 : for_model;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,5]:
#   Signal bitsig = '1'
#   Signal intsig = 21
#   Signal realsig = 1.635000e+01
#   Signal timesig = 5 ns
#   Signal stdlogicsig = 'H'
#   Signal bitarr = "0110"
#   Signal stdlogicarr = "01LH"
#   Signal intarr = (10, 11, 12)
#   Signal realarr = (1.160000e+01, 1.012200e+02)
#   Signal timearr = (15 ns, 6 ns)
#   Signal rec = ('0', 1, 3.700000e+00, 'H', "1001")
# Time [0,10]:
#   Signal bitsig = '0'
#   Signal intsig = 22
#   Signal realsig = 1.745000e+01
#   Signal timesig = 6 ns
#   Signal stdlogicsig = '-'
#   Signal bitarr = "1001"
#   Signal stdlogicarr = "1ZH-"
#   Signal intarr = (11, 12, 13)
#   Signal realarr = (1.270000e+01, 1.023200e+02)
#   Signal timearr = (16 ns, 7 ns)
#   Signal rec = ('1', 2, 4.800000e+00, '-', "0110")
# Time [0,15]:
#   Signal bitsig = '1'
#   Signal intsig = 23
#   Signal realsig = 1.855000e+01
#   Signal timesig = 7 ns
#   Signal stdlogicsig = 'U'
#   Signal bitarr = "0110"
#   Signal stdlogicarr = "ZW-U"
#   Signal intarr = (12, 13, 14)
#   Signal realarr = (1.380000e+01, 1.034200e+02)
#   Signal timearr = (17 ns, 8 ns)
#   Signal rec = ('0', 3, 5.900000e+00, 'U', "1001")
VSIM 2> quit
```

mti_SetVarValue()

Sets the value of a VHDL variable, but not SystemC variables.

Syntax

mti_SetVarValue(variable_id, value)

Arguments

Name	Type	Description
variable_id	mtiVariableIdT	A handle to a VHDL variable
value	long/void *	For a variable of scalar type, the value to be set; for a variable of real, time, or array type, a pointer to the value to be set

Return Values

Nothing

Description

mti_SetVarValue() sets the specified VHDL variable to the specified value immediately. If the variable is of type array, real, or time, then the value type is considered to be "void *" instead of "long".

You cannot use mti_SetVarValue() to set the value of a variable of type record, but you can use it to set the values of the individual scalar or array subelements.

Examples

FLI code

```c
#include <mti.h>

typedef struct varInfoT_tag {
  struct varInfoT_tag  * next;
  char                 * name;
  mtiVariableIdT         varid;
  mtiTypeIdT             typeid;
} varInfoT;

typedef struct {
  varInfoT      * var_info;    /* List of variables. */
  mtiProcessIdT   proc;        /* Test process id. */
} instanceInfoT;

static void setVarValue( mtiVariableIdT varid, mtiTypeIdT vartype )
{
  switch ( mti_GetTypeKind( vartype ) ) {
    case MTI_TYPE_ENUM:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetVarValue( varid );
        scalar_val++;
        if (( scalar_val < mti_TickLow( vartype )) ||
          ( scalar_val > mti_TickHigh( vartype ))) {
          scalar_val = mti_TickLeft( vartype );
        }
        mti_SetVarValue( varid, (long)scalar_val );
      }
      break;
    case MTI_TYPE_PHYSICAL:
    case MTI_TYPE_SCALAR:
      {
        mtiInt32T scalar_val;
        scalar_val = mti_GetVarValue( varid );
        scalar_val++;
        mti_SetVarValue( varid, (long)scalar_val );
      }
      break;
    case MTI_TYPE_ARRAY:
      {
        int             i;
        mtiTypeIdT       elem_type;
        mtiVariableIdT * elem_list;
        elem_type = mti_GetArrayElementType( vartype );
        switch ( mti_GetTypeKind( elem_type ) ) {
          case MTI_TYPE_SCALAR:
          case MTI_TYPE_PHYSICAL:
            {
              mtiInt32T * array_val = mti_GetArrayVarValue( varid, 0 );
              for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
                array_val[i]++;
              }
              mti_SetVarValue( varid, (long)array_val );
            }
```

```
        break;
      case MTI_TYPE_ARRAY:
      case MTI_TYPE_RECORD:
      default:
        elem_list = mti_GetVarSubelements( varid, 0 );
        for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
          setVarValue( elem_list[i], mti_GetVarType( elem_list[i] ));
        }
        mti_VsimFree( elem_list );
        break;
      case MTI_TYPE_ENUM:
        if ( mti_TickLength( elem_type ) <= 256 ) {
          char * array_val = mti_GetArrayVarValue( varid, 0 );
          for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
            array_val[i]++;
            if (( array_val[i] < mti_TickLow( elem_type )) ||
              ( array_val[i] > mti_TickHigh( elem_type ))) {
              array_val[i] = mti_TickLeft( elem_type );
            }
          }
          mti_SetVarValue( varid, (long)array_val );
        } else {
          mtiInt32T * array_val = mti_GetArrayVarValue( varid, 0 );
          for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
            array_val[i]++;
            if (( array_val[i] < mti_TickLow( elem_type )) ||
              ( array_val[i] > mti_TickHigh( elem_type ))) {
              array_val[i] = mti_TickLeft( elem_type );
            }
          }
          mti_SetVarValue( varid, (long)array_val );
        }
        break;
      case MTI_TYPE_REAL:
        {
          double * array_val = mti_GetArrayVarValue( varid, 0 );
          for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
            array_val[i] = array_val[i] + 1.1;
          }
          mti_SetVarValue( varid, (long)array_val );
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T * array_val = mti_GetArrayVarValue( varid, 0 );
          for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
            MTI_TIME64_ASGN( array_val[i],
                            MTI_TIME64_HI32(array_val[i]),
                            MTI_TIME64_LO32(array_val[i]) + 1 );
          }
          mti_SetVarValue( varid, (long)array_val );
        }
        break;
      }
    }
    break;
  case MTI_TYPE_RECORD:
    {
```

```
            int              i;
            mtiVariableIdT * elem_list;
            elem_list = mti_GetVarSubelements( varid, 0 );
            for ( i = 0; i < mti_TickLength( vartype ); i++ ) {
              setVarValue( elem_list[i], mti_GetVarType( elem_list[i] ));
            }
            mti_VsimFree( elem_list );
          }
          break;
      case MTI_TYPE_REAL:
        {
          double real_val;
          mti_GetVarValueIndirect( varid, &real_val );
          real_val += 1.1;
          mti_SetVarValue( varid, (long)(&real_val) );
        }
        break;
      case MTI_TYPE_TIME:
        {
          mtiTime64T time_val;
          mti_GetVarValueIndirect( varid, &time_val );
          MTI_TIME64_ASGN( time_val, MTI_TIME64_HI32(time_val),
                           MTI_TIME64_LO32(time_val) + 1 );
          mti_SetVarValue( varid, (long)(&time_val) );
        }
        break;
      default:
        break;
    }
  }

  static void checkValues( void *inst_info )
  {
    instanceInfoT *inst_data = (instanceInfoT *)inst_info;
    varInfoT      *varinfo;

    mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

    for ( varinfo = inst_data->var_info; varinfo; varinfo = varinfo->next ) {
      mti_PrintFormatted( "  Variable %s = %s:\n", varinfo->name,
                          mti_GetVarImageById( varinfo->varid ));
      setVarValue( varinfo->varid, varinfo->typeid );
    }

    mti_ScheduleWakeup( inst_data->proc, 5 );
  }

  static varInfoT * setupVariable( mtiVariableIdT varid )
  {
    varInfoT * varinfo;

    varinfo           = (varInfoT *) mti_Malloc( sizeof(varInfoT) );
    varinfo->varid    = varid;
    varinfo->name     = mti_GetVarName( varid );
    varinfo->typeid   = mti_GetVarType( varid );
    varinfo->next     = 0;

    return( varinfo );
```

```
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiProcessIdT   procid;
  mtiVariableIdT  varid;
  varInfoT      * curr_info;
  varInfoT      * varinfo;

  inst_data          = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->var_info = 0;

  for ( procid = mti_FirstProcess( mti_GetTopRegion() );
        procid; procid = mti_NextProcess() ) {
    for ( varid = mti_FirstVar( procid ); varid; varid = mti_NextVar() ) {
      varinfo = setupVariable( varid );
      if ( inst_data->var_info == 0 ) {
        inst_data->var_info = varinfo;
      }
      else {
        curr_info->next = varinfo;
      }
      curr_info = varinfo;
    }
  }

  inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                       (void *)inst_data );
  mti_ScheduleWakeup( inst_data->proc, 5 );
}

void initForeign(
  mtiRegionIdT      region,    /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char             *param,     /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 )     of integer;
  type realarray is array( 1 to 2 )     of real;
  type timearray is array( -1 to 0 )    of time;

  type rectype is record
    a : bit;
    b : integer;
    c : real;
    d : std_logic;
    e : bitarray;
  end record;

end top;

architecture a of top is

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  p1 : process

    variable bitsig      : bit        := '1';
    variable intsig      : integer    := 21;
    variable realsig     : real       := 16.35;
    variable timesig     : time       := 5 ns;
    variable stdlogicsig : std_logic := 'H';

    variable bitarr      : bitarray  := "0110";
    variable stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";
    variable intarr      : intarray  := ( 10, 11, 12 );
    variable realarr     : realarray := ( 11.6, 101.22 );
    variable timearr     : timearray := ( 15 ns, 6 ns );

    variable rec         : rectype   := ( '0', 1, 3.7, 'H', "1001" );

  begin
```

```
    wait;

  end process;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 15
# Time [0,5]:
#   Variable bitsig = '1':
#   Variable intsig = 21:
#   Variable realsig = 1.635000e+01:
#   Variable timesig = 5 ns:
#   Variable stdlogicsig = 'H':
#   Variable bitarr = "0110":
#   Variable stdlogicarr = "01LH":
#   Variable intarr = (10, 11, 12):
#   Variable realarr = (1.160000e+01, 1.012200e+02):
#   Variable timearr = (15 ns, 6 ns):
#   Variable rec = ('0', 1, 3.700000e+00, 'H', "1001"):
# Time [0,10]:
#   Variable bitsig = '0':
#   Variable intsig = 22:
#   Variable realsig = 1.745000e+01:
#   Variable timesig = 6 ns:
#   Variable stdlogicsig = '-':
#   Variable bitarr = "1001":
#   Variable stdlogicarr = "1ZH-":
#   Variable intarr = (11, 12, 13):
#   Variable realarr = (1.270000e+01, 1.023200e+02):
#   Variable timearr = (16 ns, 7 ns):
#   Variable rec = ('1', 2, 4.800000e+00, '-', "0110"):
# Time [0,15]:
#   Variable bitsig = '1':
#   Variable intsig = 23:
#   Variable realsig = 1.855000e+01:
#   Variable timesig = 7 ns:
#   Variable stdlogicsig = 'U':
#   Variable bitarr = "0110":
#   Variable stdlogicarr = "ZW-U":
#   Variable intarr = (12, 13, 14):
#   Variable realarr = (1.380000e+01, 1.034200e+02):
#   Variable timearr = (17 ns, 8 ns):
#   Variable rec = ('0', 3, 5.900000e+00, 'U', "1001"):
VSIM 2> quit
```

mti_SignalImage()

Gets the string image of a signal's value.

Syntax

value = mti_SignalImage(signal_id)

Arguments

Name	Type	Description
signal_id	mtiSignalIdT	A handle to a VHDL or SystemC signal

Return Values

Name	Type	Description
value	char *	A string image of the specified signal's value

Description

mti_SignalImage() returns a pointer to a static buffer containing the string image of the value of the specified signal. The image is the same as would be returned by the VHDL attribute 'IMAGE. The returned string is valid only until the next call to any FLI function. You must not free this pointer.

Examples

FLI code

```c
#include <mti.h>

typedef struct signalInfoT_tag {
  struct signalInfoT_tag * next;
  char                   * name;
  mtiSignalIdT             sigid;
  mtiTypeIdT               typeid;
} signalInfoT;

typedef struct {
  signalInfoT   * sig_info;     /* List of signals. */
  mtiProcessIdT   proc;         /* Test process id. */
} instanceInfoT;

static void checkValues( void *inst_info )
{
  instanceInfoT *inst_data = (instanceInfoT *)inst_info;
  signalInfoT   *siginfo;

  mti_PrintFormatted( "Time [%d,%d]:\n", mti_NowUpper(), mti_Now() );

  for ( siginfo = inst_data->sig_info; siginfo; siginfo = siginfo->next ) {
    mti_PrintFormatted( "  Signal %s = %s\n", siginfo->name,
                        mti_SignalImage( siginfo->sigid ) );
  }

  mti_ScheduleWakeup( inst_data->proc, 5 );
}

static signalInfoT * setupSignal( mtiSignalIdT sigid )
{
  signalInfoT * siginfo;

  siginfo         = (signalInfoT *) mti_Malloc( sizeof(signalInfoT) );
  siginfo->sigid  = sigid;
  siginfo->name   = mti_GetSignalNameIndirect( sigid, 0, 0 );
  siginfo->typeid = mti_GetSignalType( sigid );
  siginfo->next   = 0;

  return( siginfo );
}

static void initInstance( void * param )
{
  instanceInfoT * inst_data;
  mtiSignalIdT    sigid;
  signalInfoT   * curr_info;
  signalInfoT   * siginfo;

  inst_data           = mti_Malloc( sizeof(instanceInfoT) );
  inst_data->sig_info = 0;

  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
```

```
        siginfo = setupSignal( sigid );
        if ( inst_data->sig_info == 0 ) {
          inst_data->sig_info = siginfo;
        }
        else {
          curr_info->next = siginfo;
        }
        curr_info = siginfo;
      }

      inst_data->proc = mti_CreateProcess( "Test Process", checkValues,
                                           (void *)inst_data );
      mti_ScheduleWakeup( inst_data->proc, 6 );
    }

    void initForeign(
      mtiRegionIdT      region,   /* The ID of the region in which this    */
                                  /* foreign architecture is instantiated. */
      char             *param,    /* The last part of the string in the    */
                                  /* foreign attribute.                    */
      mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
    )
    {
      mti_AddLoadDoneCB( initInstance, 0 );
    }
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal stdlogicarr : std_logic_vector( 1 to 4 ) := "01LH";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;

  bitsig      <= not bitsig after 5 ns;
  intsig      <= intsig + 1 after 5 ns;
```

```
        physsig     <= physsig + 1 hour after 5 ns;
        realsig     <= realsig + 1.1 after 5 ns;
        timesig     <= timesig + 2 ns after 5 ns;
        stdlogicsig <= not stdlogicsig after 5 ns;

        stdlogicarr <= not stdlogicarr after 5 ns;

        rec.a       <= not rec.a after 5 ns;
        rec.b       <= rec.b + 1 after 5 ns;
        rec.c       <= not rec.c after 5 ns;

    end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.7

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.top(a)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    VSIM 1> run 18
    # Time [0,6]:
    #   Signal bitsig = '0'
    #   Signal intsig = 43
    #   Signal physsig = 4 hour
    #   Signal realsig = 1.130000e+01
    #   Signal timesig = 5 ns
    #   Signal stdlogicsig = '0'
    #   Signal stdlogicarr = "1010"
    #   Signal rec = ('1', 1, "0110")
    # Time [0,11]:
    #   Signal bitsig = '1'
    #   Signal intsig = 44
    #   Signal physsig = 5 hour
    #   Signal realsig = 1.240000e+01
    #   Signal timesig = 7 ns
    #   Signal stdlogicsig = '1'
    #   Signal stdlogicarr = "0101"
    #   Signal rec = ('0', 2, "1001")
    # Time [0,16]:
    #   Signal bitsig = '0'
    #   Signal intsig = 45
    #   Signal physsig = 6 hour
    #   Signal realsig = 1.350000e+01
    #   Signal timesig = 9 ns
    #   Signal stdlogicsig = '0'
    #   Signal stdlogicarr = "1010"
    #   Signal rec = ('1', 3, "0110")
    VSIM 2> quit
```

mti_SignalIsResolved()

Indicates whether or not the specified signal is resolved.

Syntax

resolved = mti_SignalIsResolved(signal)

Arguments

Name	Type	Description
mtiSignalIdT	signal	A handle to a VHDL signal

Return Values

Name	Type	Description
resolved	int	1 if the signal is resolved; 0 otherwise

Description

mti_SignalIsResolved returns a 1, meaning a signal is considered to be resolved, if the signal meets one of the following criteria:

- The signal is of a resolved type (for example, std_logic).

- The declaration of the signal includes a resolution function specification.

- The signal is a composite of an unresolved type but all of its subelements are resolved (for example, std_logic_vector).

- The signal is of an unresolved type but it is a subelement of a composite that is either of a resolved type or whose declaration contains a resolution function specification.

Examples

FLI code

```
#include <stdio.h>
#include <mti.h>

static void printSignalInfo( char * name )
{
    char *      signame;
    int         resolved;
    mtiSignalIdT sigid;

    sigid = mti_FindSignal( name );
    if ( sigid ) {
        signame = mti_GetSignalNameIndirect( sigid, 0, 0 );
        resolved = mti_SignalIsResolved( sigid );
        mti_PrintFormatted( "Signal %s is %sresolved.\n",
                             signame, resolved ? "" : "not " );
        mti_VsimFree( signame );
    } else {
        mti_PrintFormatted( "Signal '%s' not found.\n" );
    }
}

static void loadDoneCB( void * param )
{
    /* Unresolved scalars */
    printSignalInfo( "/top/bitsig1" );
    printSignalInfo( "/top/intsig1" );
    printSignalInfo( "/top/realsig1" );
    printSignalInfo( "/top/timesig1" );
    printSignalInfo( "/top/physsig1" );
    printSignalInfo( "/top/stdulogicsig1" );

    /* Scalars with resolved types */
    printSignalInfo( "/top/resbitsig1" );
    printSignalInfo( "/top/resintsig1" );
    printSignalInfo( "/top/resrealsig1" );
    printSignalInfo( "/top/restimesig1" );
    printSignalInfo( "/top/resphyssig1" );
    printSignalInfo( "/top/stdlogicsig1" );

    /* Resolved scalars with unresolved types */
    printSignalInfo( "/top/bitsigr" );
    printSignalInfo( "/top/intsigr" );
    printSignalInfo( "/top/realsigr" );
    printSignalInfo( "/top/timesigr" );
    printSignalInfo( "/top/physsigr" );
    printSignalInfo( "/top/stdulogicsigr" );

    /* Unresolved 1D arrays */
    printSignalInfo( "/top/bitarr1" );
    printSignalInfo( "/top/intarr1" );
    printSignalInfo( "/top/realarr1" );
    printSignalInfo( "/top/timearr1" );
    printSignalInfo( "/top/physarr1" );
    printSignalInfo( "/top/stdulogicarr1" );
```

```
/* Elements of unresolved 1D arrays */
printSignalInfo( "/top/bitarr1(3)" );
printSignalInfo( "/top/intarr1(2)" );
printSignalInfo( "/top/realarr1(-3)" );
printSignalInfo( "/top/timearr1(0)" );
printSignalInfo( "/top/physarr1(1)" );

/* 1D Arrays of resolved subelements */
printSignalInfo( "/top/rbitarr1" );
printSignalInfo( "/top/rintarr1" );
printSignalInfo( "/top/rrealarr1" );
printSignalInfo( "/top/rtimearr1" );
printSignalInfo( "/top/rphysarr1" );
printSignalInfo( "/top/stdlogicarr1" );

/* Elements of arrays of resolved subelements */
printSignalInfo( "/top/rbitarr1(6)" );
printSignalInfo( "/top/rintarr1(4)" );
printSignalInfo( "/top/rrealarr1(-1)" );
printSignalInfo( "/top/rtimearr1(1)" );
printSignalInfo( "/top/rphysarr1(3)" );
printSignalInfo( "/top/stdlogicarr1(1)" );

/* Unresolved records */
printSignalInfo( "/top/rec1" );
printSignalInfo( "/top/rec1.a" );
printSignalInfo( "/top/rec1.b" );
printSignalInfo( "/top/rec1.c" );
printSignalInfo( "/top/rec1.d" );
printSignalInfo( "/top/rec1.e" );
printSignalInfo( "/top/rec1.f" );
printSignalInfo( "/top/rec1.g" );

/* Records of resolved elements */
printSignalInfo( "/top/rec2" );
printSignalInfo( "/top/rec2.b" );
printSignalInfo( "/top/rec2.i" );
printSignalInfo( "/top/rec2.r" );
printSignalInfo( "/top/rec2.t" );
printSignalInfo( "/top/rec2.s" );
printSignalInfo( "/top/rec2.p" );

/* Records of mixed resolution */
printSignalInfo( "/top/rec3" );
printSignalInfo( "/top/rec3.f1" );
printSignalInfo( "/top/rec3.f2" );
printSignalInfo( "/top/rec3.f2.a" );
printSignalInfo( "/top/rec3.f2.b" );
printSignalInfo( "/top/rec3.f2.c" );
printSignalInfo( "/top/rec3.f2.d" );
printSignalInfo( "/top/rec3.f2.e" );
printSignalInfo( "/top/rec3.f2.f" );
printSignalInfo( "/top/rec3.f2.g" );
printSignalInfo( "/top/rec3.f3" );
printSignalInfo( "/top/rec3.f4" );
printSignalInfo( "/top/rec3.f4.b" );
printSignalInfo( "/top/rec3.f4.i" );
```

```
        printSignalInfo( "/top/rec3.f4.r" );
        printSignalInfo( "/top/rec3.f4.t" );
        printSignalInfo( "/top/rec3.f4.s" );
        printSignalInfo( "/top/rec3.f4.p" );
    }

    void initForeign(
      mtiRegionIdT       region,     /* The ID of the region in which this      */
                                     /* foreign architecture is instantiated.   */
      char              *param,      /* The last part of the string in the      */
                                     /* foreign attribute.                      */
      mtiInterfaceListT *generics,   /* A list of generics for the foreign model.*/
      mtiInterfaceListT *ports       /* A list of ports for the foreign model.  */
    )
    {
        mti_AddLoadDoneCB( loadDoneCB, 0 );
    }
```

HDL code

```vhdl
library ieee;
use ieee.std_logic_1164.all;

package typepkg is

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

  type int_vector  is array (natural range <>) of integer;
  type real_vector is array (natural range <>) of real;
  type time_vector is array (natural range <>) of time;
  type phys_vector is array (natural range <>) of bigtime;

  FUNCTION resolve_bit  ( s : bit_vector )  RETURN bit;
  FUNCTION resolve_int  ( s : int_vector )  RETURN integer;
  FUNCTION resolve_real ( s : real_vector ) RETURN real;
  FUNCTION resolve_time ( s : time_vector ) RETURN time;
  FUNCTION resolve_phys ( s : phys_vector ) RETURN bigtime;

  type bitarray  is array( 3 downto 0 ) of bit;
  type intarray  is array( 1 to 3 ) of integer;
  type realarray is array( -3 to -1 ) of real;
  type timearray is array( 0 to 2 ) of time;
  type hourarray is array( 1 to 2 ) of bigtime;

  subtype resbit  is resolve_bit  bit;
  subtype resint  is resolve_int  integer;
  subtype resreal is resolve_real real;
  subtype restime is resolve_time time;
  subtype resphys is resolve_phys bigtime;

  type rectype1 is record
    a : bit;
    b : integer;
    c : real;
    d : time;
    e : std_logic;
    f : bigtime;
    g : std_ulogic;
  end record;

  type rectype2 is record
    b : resbit;
    i : resint;
    r : resreal;
    t : restime;
    s : std_logic;
    p : resphys;
  end record;
```

```
      type rectype3 is record
        f1 : resbit;
        f2 : rectype1;
        f3 : integer;
        f4 : rectype2;
      end record;

      type rbitarray is array( 7 downto 0 ) of resbit;
      type rintarray is array( 2 to 4 ) of resint;
      type rrealarray is array( 0 downto -2 ) of resreal;
      type rtimearray is array( 1 to 3 ) of restime;
      type rhourarray is array( 1 to 3 ) of resphys;

  end package typepkg;

  package body typepkg is

    FUNCTION resolve_bit ( s : bit_vector ) RETURN bit IS
      VARIABLE result : bit := '0';
    BEGIN
      IF (s'LENGTH = 1) THEN
        RETURN s(s'LOW);
      ELSE
        FOR i IN s'RANGE LOOP
          if ( s(i) = '1' ) then
            result := '1';
          end if;
        END LOOP;
      END IF;
      RETURN result;
    END resolve_bit;

    FUNCTION resolve_int ( s : int_vector ) RETURN integer IS
      VARIABLE result : integer := 0;
    BEGIN
      IF (s'LENGTH = 1) THEN
        RETURN s(s'LOW);
      ELSE
        FOR i IN s'RANGE LOOP
          result := result + s(i);
        END LOOP;
      END IF;
      RETURN result;
    END resolve_int;

    FUNCTION resolve_real ( s : real_vector ) RETURN real IS
      VARIABLE result : real := 0.0;
    BEGIN
      IF (s'LENGTH = 1) THEN
        RETURN s(s'LOW);
      ELSE
        FOR i IN s'RANGE LOOP
          result := result + s(i);
        END LOOP;
      END IF;
      RETURN result;
    END resolve_real;
```

```
    FUNCTION resolve_time ( s : time_vector ) RETURN time IS
      VARIABLE result : time := 0 ns;
    BEGIN
      IF (s'LENGTH = 1) THEN
        RETURN s(s'LOW);
      ELSE
        FOR i IN s'RANGE LOOP
          result := result + s(i);
        END LOOP;
      END IF;
      RETURN result;
    END resolve_time;

    FUNCTION resolve_phys ( s : phys_vector ) RETURN bigtime IS
      VARIABLE result : bigtime := 0 hour;
    BEGIN
      IF (s'LENGTH = 1) THEN
        RETURN s(s'LOW);
      ELSE
        FOR i IN s'RANGE LOOP
          result := result + s(i);
        END LOOP;
      END IF;
      RETURN result;
    END resolve_phys;

end package body typepkg;

library ieee;
use ieee.std_logic_1164.all;

use work.typepkg.all;

entity top is
end top;

architecture a of top is

-- Unresolved scalars
  signal bitsig1       : bit         := '1';
  signal intsig1       : integer     := 21;
  signal realsig1      : real        := 21.21;
  signal timesig1      : time        := 21 ns;
  signal physsig1      : bigtime     := 21 hour;
  signal stdulogicsig1 : std_ulogic := 'L';

-- Scalars with resolved types
  signal resbitsig1    : resbit    := '0';
  signal resintsig1    : resint    := 42;
  signal resrealsig1   : resreal   := 11.9;
  signal restimesig1   : restime   := 64 ns;
  signal resphyssig1   : resphys   := 1 day;
  signal stdlogicsig1  : std_logic := 'H';

-- Resolved scalars with unresolved types
  signal bitsigr       : resolve_bit  bit        := '1';
  signal intsigr       : resolve_int  integer    := 17;
  signal realsigr      : resolve_real real       := 6.25;
```

```
    signal timesigr       : resolve_time time       := 2 ns;
    signal physsigr       : resolve_phys bigtime    := 2 week;
    signal stdulogicsigr : resolved      std_ulogic := '1';

 -- Unresolved 1D arrays
    signal bitarr1        : bitarray := "0110";
    signal intarr1        : intarray := ( 10, 11, 12 );
    signal realarr1       : realarray := ( 7.7, 3.2, -8.1 );
    signal timearr1       : timearray := ( 4 ns, 5 ns, 6 ns );
    signal physarr1       : hourarray := ( 40 hour, 50 hour );
    signal stdulogicarr1 : std_ulogic_vector( 3 downto 0 ) := "HL01";

 -- 1D Arrays of resolved subelements
    signal rbitarr1       : rbitarray := "10110110";
    signal rintarr1       : rintarray := ( 30, 41, 52 );
    signal rrealarr1      : rrealarray := ( 17.6, -43.8, 9.1 );
    signal rtimearr1      : rtimearray := ( 1 ns, 3 ns, 5 ns );
    signal rphysarr1      : rhourarray := ( 1 day, 10 hour, 2 week );
    signal stdlogicarr1  : std_logic_vector( 1 to 4 ) := "-X0U";

 -- Unresolved records
    signal rec1           : rectype1 := ( '0', 1, 1.1, 1 ns, 'X', 1 hour, 'L' );

 -- Records of resolved elements
    signal rec2           : rectype2 := ( '1', 5, 2.01, 3 ns, 'H', 2 hour );

 -- Records of mixed resolution
    signal rec3           : rectype3 := ( '1',
                                        ( '1', 4, 8.5, 19 ns, 'L', 1 day, 'Z' ),
                                         168,
                                        ( '0', 81, 6.25, 7 ns, '1', 4 hour )
                                        );

 begin

   bitsig1 <= not bitsig1 after 5  ns;

 end a;
```

Simulation output

```
% vsim -c top -foreign "initForeign for_model.sl"
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.7c

# vsim -foreign {initForeign for_model.sl} -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.typepkg(body)
# Loading work.top(a)
# Loading ./for_model.sl
# Signal bitsig1 is not resolved.
# Signal intsig1 is not resolved.
# Signal realsig1 is not resolved.
# Signal timesig1 is not resolved.
# Signal physsig1 is not resolved.
# Signal stdulogicsig1 is not resolved.
# Signal resbitsig1 is resolved.
# Signal resintsig1 is resolved.
# Signal resrealsig1 is resolved.
# Signal restimesig1 is resolved.
# Signal resphyssig1 is resolved.
# Signal stdlogicsig1 is resolved.
# Signal bitsigr is resolved.
# Signal intsigr is resolved.
# Signal realsigr is resolved.
# Signal timesigr is resolved.
# Signal physsigr is resolved.
# Signal stdulogicsigr is resolved.
# Signal bitarr1 is not resolved.
# Signal intarr1 is not resolved.
# Signal realarr1 is not resolved.
# Signal timearr1 is not resolved.
# Signal physarr1 is not resolved.
# Signal stdulogicarr1 is not resolved.
# Signal bitarr1(3) is not resolved.
# Signal intarr1(2) is not resolved.
# Signal realarr1(-3) is not resolved.
# Signal timearr1(0) is not resolved.
# Signal physarr1(1) is not resolved.
# Signal rbitarr1 is resolved.
# Signal rintarr1 is resolved.
# Signal rrealarr1 is resolved.
# Signal rtimearr1 is resolved.
# Signal rphysarr1 is resolved.
# Signal stdlogicarr1 is resolved.
# Signal rbitarr1(6) is resolved.
# Signal rintarr1(4) is resolved.
# Signal rrealarr1(-1) is resolved.
# Signal rtimearr1(1) is resolved.
# Signal rphysarr1(3) is resolved.
# Signal stdlogicarr1(1) is resolved.
# Signal rec1 is not resolved.
# Signal rec1.a is not resolved.
# Signal rec1.b is not resolved.
# Signal rec1.c is not resolved.
```

```
# Signal rec1.d is not resolved.
# Signal rec1.e is resolved.
# Signal rec1.f is not resolved.
# Signal rec1.g is not resolved.
# Signal rec2 is resolved.
# Signal rec2.b is resolved.
# Signal rec2.i is resolved.
# Signal rec2.r is resolved.
# Signal rec2.t is resolved.
# Signal rec2.s is resolved.
# Signal rec2.p is resolved.
# Signal rec3 is not resolved.
# Signal rec3.f1 is resolved.
# Signal rec3.f2 is not resolved.
# Signal rec3.f2.a is not resolved.
# Signal rec3.f2.b is not resolved.
# Signal rec3.f2.c is not resolved.
# Signal rec3.f2.d is not resolved.
# Signal rec3.f2.e is resolved.
# Signal rec3.f2.f is not resolved.
# Signal rec3.f2.g is not resolved.
# Signal rec3.f3 is not resolved.
# Signal rec3.f4 is resolved.
# Signal rec3.f4.b is resolved.
# Signal rec3.f4.i is resolved.
# Signal rec3.f4.r is resolved.
# Signal rec3.f4.t is resolved.
# Signal rec3.f4.s is resolved.
# Signal rec3.f4.p is resolved.
VSIM 1> quit
```

mti_TickDir()

Gets the direction of a type.

Syntax

direction = mti_TickDir(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
direction	mtiInt32T	+1 for ascending, -1 for descending, or 0 for no direction

Description

mti_TickDir() returns the index direction of an array type or the range direction of any type that has a range.

Examples

FLI code

```c
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:    return "Scalar";
    case MTI_TYPE_ARRAY:     return "Array";
    case MTI_TYPE_RECORD:    return "Record";
    case MTI_TYPE_ENUM:      return "Enumeration";
    case MTI_TYPE_PHYSICAL:  return "Physical";
    case MTI_TYPE_REAL:      return "Real";
    case MTI_TYPE_TIME:      return "Time";
    default:                 return "UNKNOWN";
  }
}

static char * getDirStr( mtiTypeIdT typeid )
{
  switch( mti_TickDir( typeid ) ) {
    case -1:  return "Descending";
    case 0:   return "No direction";
    case 1:   return "Ascending";
    default:  return "UNKNOWN";
  }
}

static void initInstance( void * param )
{
  mtiSignalIdT sigid;
  mtiTypeIdT   typeid;

  mti_PrintMessage( "Design Signals:\n" );
  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    typeid = mti_GetSignalType( sigid );
    mti_PrintFormatted( "%14s: type %-12s; direction = %s (%d)\n",
                        mti_GetSignalName( sigid ),
                        getTypeStr( typeid ),
                        getDirStr( typeid ), mti_TickDir( typeid ) );
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( 3 downto 0 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal bitarr      : bitarray := "1100";
  signal stdlogicarr : std_logic_vector( 3 downto 0 ) := "01LH";
  signal uparray     : bit_vector( 1 to 4 ) := "0101";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
    end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.top(a)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    # Design Signals:
    #        bitsig: type Enumeration ; direction = Ascending (1)
    #        intsig: type Scalar      ; direction = Ascending (1)
    #       physsig: type Physical    ; direction = Ascending (1)
    #       realsig: type Real        ; direction = Ascending (1)
    #       timesig: type Time        ; direction = Ascending (1)
    #    stdlogicsig: type Enumeration ; direction = Ascending (1)
    #        bitarr: type Array        ; direction = Descending (-1)
    #    stdlogicarr: type Array        ; direction = Descending (-1)
    #       uparray: type Array        ; direction = Ascending (1)
    #           rec: type Record       ; direction = No direction (0)
    VSIM 1> quit
```

mti_TickHigh()

Gets the high value of a ranged type.

Syntax

high = mti_TickHigh(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
high	mtiInt32T	The high value of the range of the specified type; 0 for real, time, and record types

Description

mti_TickHigh() returns the value of type'HIGH for ranged types. For real, time, and record types, mti_TickHigh() returns 0.

Examples

FLI code

```c
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:    return "Scalar";
    case MTI_TYPE_ARRAY:     return "Array";
    case MTI_TYPE_RECORD:    return "Record";
    case MTI_TYPE_ENUM:      return "Enumeration";
    case MTI_TYPE_PHYSICAL:  return "Physical";
    case MTI_TYPE_REAL:      return "Real";
    case MTI_TYPE_TIME:      return "Time";
    default:                 return "UNKNOWN";
  }
}

static void initInstance( void * param )
{
  mtiSignalIdT sigid;
  mtiTypeIdT   typeid;

  mti_PrintMessage( "Design Signals:\n" );
  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    typeid = mti_GetSignalType( sigid );
    mti_PrintFormatted( "%14s: type %-12s; low = %d, high = %d\n",
                        mti_GetSignalName( sigid ), getTypeStr( typeid ),
                        mti_TickLow( typeid ), mti_TickHigh( typeid ));
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( -2 downto -5 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
  units
    hour;
    day   = 24 hour;
    week  = 7 day;
    month = 4 week;
    year  = 12 month;
  end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal bitarr      : bitarray := "1100";
  signal stdlogicarr : std_logic_vector( 3 downto 0 ) := "01LH";
  signal uparray     : bit_vector( 1 to 4 ) := "0101";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
      end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Design Signals:
#        bitsig: type Enumeration ; low = 0, high = 1
#        intsig: type Scalar      ; low = -2147483648, high = 2147483647
#       physsig: type Physical    ; low = 0, high = 2147483647
#       realsig: type Real        ; low = 0, high = 0
#       timesig: type Time        ; low = 0, high = 0
#    stdlogicsig: type Enumeration ; low = 0, high = 8
#        bitarr: type Array        ; low = -5, high = -2
#    stdlogicarr: type Array       ; low = 0, high = 3
#       uparray: type Array        ; low = 1, high = 4
#           rec: type Record       ; low = 0, high = 0
VSIM 1> quit
```

mti_TickLeft()

Gets the left value of a ranged type.

Syntax

left = mti_TickLeft(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
left	mtiInt32T	The left value of the range of the specified type; 0 for real, time, and record types

Description

mti_TickLeft() returns the value of type'LEFT for ranged types. For real, time, and record types, mti_TickLeft() returns 0.

Examples

FLI code

```
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:    return "Scalar";
    case MTI_TYPE_ARRAY:     return "Array";
    case MTI_TYPE_RECORD:    return "Record";
    case MTI_TYPE_ENUM:      return "Enumeration";
    case MTI_TYPE_PHYSICAL:  return "Physical";
    case MTI_TYPE_REAL:      return "Real";
    case MTI_TYPE_TIME:      return "Time";
    default:                 return "UNKNOWN";
  }
}

static void initInstance( void * param )
{
  mtiSignalIdT sigid;
  mtiTypeIdT   typeid;

  mti_PrintMessage( "Design Signals:\n" );
  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    typeid = mti_GetSignalType( sigid );
    mti_PrintFormatted( "%14s: type %-12s; left = %d, right = %d\n",
                        mti_GetSignalName( sigid ), getTypeStr( typeid ),
                        mti_TickLeft( typeid ), mti_TickRight( typeid ));
  }
}

void initForeign(
  mtiRegionIdT      region,    /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( -2 downto -5 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig       : bit        := '1';
  signal intsig       : integer    := 42;
  signal physsig      : bigtime    := 3 hour;
  signal realsig      : real       := 10.2;
  signal timesig      : time       := 3 ns;
  signal stdlogicsig  : std_logic  := 'H';

  signal bitarr       : bitarray := "1100";
  signal stdlogicarr  : std_logic_vector( 3 downto 0 ) := "01LH";
  signal uparray      : bit_vector( 1 to 4 ) := "0101";

  signal rec          : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
        end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.top(a)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    # Design Signals:
    #        bitsig: type Enumeration ; left = 0, right = 1
    #        intsig: type Scalar      ; left = -2147483648, right = 2147483647
    #       physsig: type Physical    ; left = 0, right = 2147483647
    #       realsig: type Real        ; left = 0, right = 0
    #       timesig: type Time        ; left = 0, right = 0
    #    stdlogicsig: type Enumeration ; left = 0, right = 8
    #        bitarr: type Array       ; left = -2, right = -5
    #    stdlogicarr: type Array       ; left = 3, right = 0
    #       uparray: type Array       ; left = 1, right = 4
    #           rec: type Record      ; left = 0, right = 0
    VSIM 1> quit
```

mti_TickLength()

Gets the length of a type.

Syntax

length = mti_TickLength(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
length	mtiInt32T	The length of the range of the specified type; the number of fields for record types; 0 for real and time types

Description

mti_TickLength() returns the value of type'LENGTH (type'HIGH - type'LOW + 1). For record types, the number of fields is returned. For SystemC types, the size of the type in bits is returned. For real and time types, 0 is returned.

For non-SystemC types, mti_TickLength() returns 0 if the length of the range is greater than will fit in a 32-bit integer.

Examples

FLI code

```
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:    return "Scalar";
    case MTI_TYPE_ARRAY:     return "Array";
    case MTI_TYPE_RECORD:    return "Record";
    case MTI_TYPE_ENUM:      return "Enumeration";
    case MTI_TYPE_PHYSICAL:  return "Physical";
    case MTI_TYPE_REAL:      return "Real";
    case MTI_TYPE_TIME:      return "Time";
    default:                 return "UNKNOWN";
  }
}

static void initInstance( void * param )
{
  mtiSignalIdT sigid;
  mtiTypeIdT   typeid;

  mti_PrintMessage( "Design Signals:\n" );
  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    typeid = mti_GetSignalType( sigid );
    mti_PrintFormatted( "%14s: type %-12s; length = %d\n",
                        mti_GetSignalName( sigid ), getTypeStr( typeid ),
                        mti_TickLength( typeid ));
  }
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this     */
                              /* foreign architecture is instantiated.  */
  char             *param,    /* The last part of the string in the     */
                              /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( -2 downto -5 ) of bit;
  type intrange is range 0 to 255;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';
  signal rangesig    : intrange  := 128;

  signal bitarr      : bitarray := "1100";
  signal stdlogicarr : std_logic_vector( 3 downto 2 ) := "01";
  signal uparray     : bit_vector( 1 to 5 ) := "01010";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin
```

```
        inst1 : for_model;

    end a;
```

Simulation output

```
    % vsim -c top
    Reading .../modeltech/tcl/vsim/pref.tcl

    # 5.4b

    # vsim -c top
    # Loading .../modeltech/sunos5/../std.standard
    # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
    # Loading work.top(a)
    # Loading work.for_model(a)
    # Loading ./for_model.sl
    # Design Signals:
    #          bitsig: type Enumeration ; length = 2
    #          intsig: type Scalar      ; length = 0
    #        physsig: type Physical    ; length = 0
    #         realsig: type Real        ; length = 0
    #         timesig: type Time        ; length = 0
    #    stdlogicsig: type Enumeration ; length = 9
    #        rangesig: type Scalar      ; length = 256
    #          bitarr: type Array       ; length = 4
    #    stdlogicarr: type Array       ; length = 2
    #        uparray: type Array       ; length = 5
    #            rec: type Record      ; length = 3
    VSIM 1> quit
```

mti_TickLow()

Gets the low value of a ranged type.

Syntax

low = mti_TickLow(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
low	mtiInt32T	The low value of the range of the specified type; 0 for real, time, and record types

Description

mti_TickLow() returns the value of type'LOW for ranged types. For real, time, and record types, mti_TickLow() returns 0.

Examples

FLI code

```
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:    return "Scalar";
    case MTI_TYPE_ARRAY:     return "Array";
    case MTI_TYPE_RECORD:    return "Record";
    case MTI_TYPE_ENUM:      return "Enumeration";
    case MTI_TYPE_PHYSICAL:  return "Physical";
    case MTI_TYPE_REAL:      return "Real";
    case MTI_TYPE_TIME:      return "Time";
    default:                 return "UNKNOWN";
  }
}

static void initInstance( void * param )
{
  mtiSignalIdT sigid;
  mtiTypeIdT   typeid;

  mti_PrintMessage( "Design Signals:\n" );
  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    typeid = mti_GetSignalType( sigid );
    mti_PrintFormatted( "%14s: type %-12s; low = %d, high = %d\n",
                    mti_GetSignalName( sigid ), getTypeStr( typeid ),
                     mti_TickLow( typeid ), mti_TickHigh( typeid ));
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( -2 downto -5 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal bitarr      : bitarray := "1100";
  signal stdlogicarr : std_logic_vector( 3 downto 0 ) := "01LH";
  signal uparray     : bit_vector( 1 to 4 ) := "0101";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
        end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# Design Signals:
#        bitsig: type Enumeration ; low = 0, high = 1
#        intsig: type Scalar      ; low = -2147483648, high = 2147483647
#       physsig: type Physical    ; low = 0, high = 2147483647
#       realsig: type Real        ; low = 0, high = 0
#       timesig: type Time        ; low = 0, high = 0
#    stdlogicsig: type Enumeration ; low = 0, high = 8
#        bitarr: type Array       ; low = -5, high = -2
#    stdlogicarr: type Array       ; low = 0, high = 3
#       uparray: type Array       ; low = 1, high = 4
#           rec: type Record      ; low = 0, high = 0
VSIM 1> quit
```

mti_TickRight()

Gets the right value of a ranged type.

Syntax

right = mti_TickRight(type_id)

Arguments

Name	Type	Description
type_id	mtiTypeIdT	A handle to a VHDL or SystemC type

Return Values

Name	Type	Description
right	mtiInt32T	The right value of the range of the specified type; 0 for real, time, and record types

Description

mti_TickRight() returns the value of type'RIGHT for ranged types. For real, time, and record types, mti_TickRight() returns 0.

Examples

FLI code

```c
#include <mti.h>

static char * getTypeStr( mtiTypeIdT typeid )
{
  switch ( mti_GetTypeKind( typeid ) ) {
    case MTI_TYPE_SCALAR:    return "Scalar";
    case MTI_TYPE_ARRAY:     return "Array";
    case MTI_TYPE_RECORD:    return "Record";
    case MTI_TYPE_ENUM:      return "Enumeration";
    case MTI_TYPE_PHYSICAL:  return "Physical";
    case MTI_TYPE_REAL:      return "Real";
    case MTI_TYPE_TIME:      return "Time";
    default:                 return "UNKNOWN";
  }
}

static void initInstance( void * param )
{
  mtiSignalIdT sigid;
  mtiTypeIdT   typeid;

  mti_PrintMessage( "Design Signals:\n" );
  for ( sigid = mti_FirstSignal( mti_GetTopRegion() );
        sigid; sigid = mti_NextSignal() ) {
    typeid = mti_GetSignalType( sigid );
    mti_PrintFormatted( "%14s: type %-12s; left = %d, right = %d\n",
                        mti_GetSignalName( sigid ), getTypeStr( typeid ),
                        mti_TickLeft( typeid ), mti_TickRight( typeid ));
  }
}

void initForeign(
  mtiRegionIdT       region,   /* The ID of the region in which this     */
                               /* foreign architecture is instantiated.  */
  char              *param,    /* The last part of the string in the     */
                               /* foreign attribute.                     */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.  */
)
{
  mti_AddLoadDoneCB( initInstance, 0 );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl;";
begin
end a;

library ieee;
use ieee.std_logic_1164.all;

entity top is

  type bitarray is array( -2 downto -5 ) of bit;

  type rectype is record
    a : bit;
    b : integer;
    c : bitarray;
  end record;

  type bigtime is range 0 to integer'high
    units
      hour;
      day   = 24 hour;
      week  = 7 day;
      month = 4 week;
      year  = 12 month;
    end units;

end top;

architecture a of top is

  signal bitsig      : bit       := '1';
  signal intsig      : integer   := 42;
  signal physsig     : bigtime   := 3 hour;
  signal realsig     : real      := 10.2;
  signal timesig     : time      := 3 ns;
  signal stdlogicsig : std_logic := 'H';

  signal bitarr      : bitarray := "1100";
  signal stdlogicarr : std_logic_vector( 3 downto 0 ) := "01LH";
  signal uparray     : bit_vector( 1 to 4 ) := "0101";

  signal rec         : rectype   := ( '0', 0, "1001" );

  component for_model
  end component;

  for all : for_model use entity work.for_model(a);

begin

  inst1 : for_model;
```

```
      end a;
```

Simulation output

```
      % vsim -c top
      Reading .../modeltech/tcl/vsim/pref.tcl

      # 5.4b

      # vsim -c top
      # Loading .../modeltech/sunos5/../std.standard
      # Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
      # Loading work.top(a)
      # Loading work.for_model(a)
      # Loading ./for_model.sl
      # Design Signals:
      #        bitsig: type Enumeration ; left = 0, right = 1
      #        intsig: type Scalar      ; left = -2147483648, right = 2147483647
      #       physsig: type Physical    ; left = 0, right = 2147483647
      #       realsig: type Real        ; left = 0, right = 0
      #       timesig: type Time        ; left = 0, right = 0
      #    stdlogicsig: type Enumeration ; left = 0, right = 8
      #        bitarr: type Array       ; left = -2, right = -5
      #    stdlogicarr: type Array       ; left = 3, right = 0
      #       uparray: type Array       ; left = 1, right = 4
      #           rec: type Record      ; left = 0, right = 0
      VSIM 1> quit
```

mti_TimeToString()

Returns a conversion of a specified time value, formatted according to specified flags.

Syntax

value = mti_TimeToString(time_value, flags)

Arguments

Name	Type	Description
time_value	mtiTime64T *	The time value to be converted
flags	mtiTimeFlagT	The type of formatting for the specified time value.

Return Values

Name	Type	Description
value	char *	A version of your specified time, formatted based on your settings.

Description

mti_NowFormatted() returns the current simulation time using the settings as specified in the function. The return value is stored in a buffer and should be used immediately.

You can specify any number of flag arguments in a pipe (|) separated list.

The argument flag can include any of the following:

MTI_TIME_BEST_UNITS	Determines the best unit to use for display.
MTI_TIME_INSERT_COMMAS	Inserts commas every three digits.
MTI_TIME_NO_DEF_UNIT	Does not display the default units.
MTI_TIME_FREQUENCY	Displays the time as 1/time in hz.

mti_VsimFree()

Frees simulator-allocated memory.

Syntax

mti_VsimFree(pointer)

Arguments

Name	Type	Description
pointer	void *	A pointer to memory previously allocated with malloc() by an FLI function

Return Values

Nothing

Description

mti_VsimFree() returns the specified block of memory allocated with malloc() by an FLI function to the general memory pool.

You cannot use mti_VsimFree() for memory allocated by calls to mti_Malloc() nor for memory allocated with malloc() by a user-written application. The documentation for each FLI function that allocates memory indicates whether that memory should be freed with mti_Free() or mti_VsimFree() or whether it should not be freed.

Examples

FLI code

```
#include <stdlib.h>
#include <mti.h>

typedef enum {
  STD_LOGIC_U,
  STD_LOGIC_X,
  STD_LOGIC_0,
  STD_LOGIC_1,
  STD_LOGIC_Z,
  STD_LOGIC_W,
  STD_LOGIC_L,
  STD_LOGIC_H,
  STD_LOGIC_D
} standardLogicType;

typedef struct {
  mtiSignalIdT  sigid;
  mtiProcessIdT procid;
} instanceInfoT;

char * convertStdLogicValue( mtiInt32T sigval )
{
  char * retval;

  switch ( sigval ) {
    case STD_LOGIC_U:  retval = "'U'";  break;
    case STD_LOGIC_X:  retval = "'X'";  break;
    case STD_LOGIC_0:  retval = "'0'";  break;
    case STD_LOGIC_1:  retval = "'1'";  break;
    case STD_LOGIC_Z:  retval = "'Z'";  break;
    case STD_LOGIC_W:  retval = "'W'";  break;
    case STD_LOGIC_L:  retval = "'L'";  break;
    case STD_LOGIC_H:  retval = "'H'";  break;
    case STD_LOGIC_D:  retval = "'-'";  break;
    default:  retval = "?";  break;
  }
  return retval;
}

void watchSignal( void * param )
{
  char          * region_name;
  instanceInfoT * inst = (instanceInfoT*)param;
  mtiInt32T       sigval;

  region_name = mti_GetRegionFullName( mti_GetSignalRegion(inst->sigid) );
  sigval      = mti_GetSignalValue( inst->sigid );
  mti_PrintFormatted( "Time [%d,%d] delta %d: Signal %s/%s is %s\n",
                      mti_NowUpper(), mti_Now(), mti_Delta(),
                      region_name, mti_GetSignalName( inst->sigid ),
                      convertStdLogicValue( sigval ) );

  mti_VsimFree( region_name );
```

```
    if ( mti_Now() >= 30 ) {
      mti_PrintMessage( "Turning off signal watcher.\n" );
      mti_Free( inst );
    } else {
      mti_ScheduleWakeup( inst->procid, 5 );
    }
  }

  void initForeign(
    mtiRegionIdT       region,    /* The ID of the region in which this     */
                                  /* foreign architecture is instantiated.  */
    char              *param,     /* The last part of the string in the     */
                                  /* foreign attribute.                     */
    mtiInterfaceListT *generics,  /* A list of generics for the foreign model.*/
    mtiInterfaceListT *ports      /* A list of ports for the foreign model.  */
  )
  {
    instanceInfoT * inst;

    inst         = (instanceInfoT *) mti_Malloc( sizeof(instanceInfoT) );
    inst->sigid  = mti_FindSignal( "/top/s1" );
    inst->procid = mti_CreateProcess( "sigWatcher", watchSignal, inst );
  }
```

HDL code

```
  entity for_model is
  end for_model;

  architecture a of for_model is
    attribute foreign of a : architecture is "initForeign for_model.sl";
  begin
  end a;

  library ieee;
  use ieee.std_logic_1164.all;

  entity top is
  end top;

  architecture a of top is

    signal s1 : std_logic := '0';

    component for_model is
    end component;

    for all : for_model use entity work.for_model(a);

  begin

    s1 <= not s1 after 5 ns;

    i1 : for_model;

  end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading .../modeltech/sunos5/../ieee.std_logic_1164(body)
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
VSIM 1> run 50
# Time [0,0] delta 0: Signal /top/s1 is '0'
# Time [0,5] delta 0: Signal /top/s1 is '1'
# Time [0,10] delta 0: Signal /top/s1 is '0'
# Time [0,15] delta 0: Signal /top/s1 is '1'
# Time [0,20] delta 0: Signal /top/s1 is '0'
# Time [0,25] delta 0: Signal /top/s1 is '1'
# Time [0,30] delta 0: Signal /top/s1 is '0'
# Turning off signal watcher.
VSIM 2> quit
```

mti_WriteProjectEntry()

Writes an entry into the project (.ini) file.

Syntax

mti_WriteProjectEntry(key, value)

Arguments

Name	Type	Description
key	char *	A string containing a keyword
value	char *	A string containing the value of the keyword

Return Values

Nothing

Description

mti_WriteProjectEntry() writes an entry into the *modelsim.ini* project file in the form:

```
key = value
```

The FLI writes the new entry at the end of the [vsim] section.

Examples

FLI code

```
#include <mti.h>

void loadDoneCallback( void * param )
{
  char * entry;

  entry = mti_FindProjectEntry( "vsim", "MyConfig", 0 );
  mti_PrintFormatted( "MyConfig = %s\n", entry );
  mti_VsimFree( entry );

  entry = mti_FindProjectEntry( "vsim", "MyDesign", 0 );
  mti_PrintFormatted( "MyDesign = %s\n", entry );
  mti_VsimFree( entry );

  entry = mti_FindProjectEntry( "vsim", "MyMemory", 0 );
  mti_PrintFormatted( "MyMemory = %s\n", entry );
  mti_VsimFree( entry );
}

void initForeign(
  mtiRegionIdT      region,   /* The ID of the region in which this    */
                              /* foreign architecture is instantiated. */
  char             *param,    /* The last part of the string in the    */
                              /* foreign attribute.                    */
  mtiInterfaceListT *generics, /* A list of generics for the foreign model.*/
  mtiInterfaceListT *ports     /* A list of ports for the foreign model.   */
)
{
  mti_AddLoadDoneCB( loadDoneCallback, 0 );
  mti_WriteProjectEntry( "MyConfig", "Solaris" );
  mti_WriteProjectEntry( "MyDesign", "cpu" );
  mti_WriteProjectEntry( "MyMemory", "4Meg" );
}
```

HDL code

```
entity for_model is
end for_model;

architecture a of for_model is
  attribute foreign of a : architecture is "initForeign for_model.sl";
begin
end a;

entity top is
end top;

architecture a of top is

  signal s1 : bit := '0';

  component for_model is
  end component;

  for all : for_model use entity work.for_model(a);

begin

  i1 : for_model;

  s1 <= not s1 after 5 ns;

end a;
```

Simulation output

```
% vsim -c top
Reading .../modeltech/tcl/vsim/pref.tcl

# 5.4b

# vsim -c top
# Loading .../modeltech/sunos5/../std.standard
# Loading work.top(a)
# Loading work.for_model(a)
# Loading ./for_model.sl
# MyConfig = Solaris
# MyDesign = cpu
# MyMemory = 4Meg
VSIM 1> quit
% grep My modelsim.ini
MyConfig = Solaris
MyDesign = cpu
MyMemory = 4Meg
```

End-User License Agreement
with EDA Software Supplemental Terms

Use of software (including any updates) and/or hardware is subject to the End-User License Agreement together with the Mentor Graphics EDA Software Supplement Terms. You can view and print a copy of this agreement at:

mentor.com/eula