System Verilog 3.1 Donation
Part III: Coverage API

Version 1.1, May 2002

SYNOPSYS®

1

Coverage API

This chapter describes an interface to the coverage engine that permits a user to obtain coverage data during simulation about any instance in the design. This interface provides such an interface to all coverage capabilities: line, condition, fsm, toggle, user-expression. Additionally, a C interface is also documented, for use by other Synopsys tools, to allow for integration of realtime coverage data. This C interface is not, at this time, been made available for use by any other than Synopsys tools.

Verilog Interface to Realtime Coverage

The interface consists of the following system tasks:

$cm_coverage. Starts and stops specific types of coverage data for this simulation, or checks what coverage data will be collected in this simulation run.

$cm_get_coverage. Accesses the current coverage totals.

$cm_get_limit. Accesses the limiting value for coverage (i.e. the value representing 100% coverage).

These functions will be described in more detail below.

Header File

To support the realtime interface, include a Verilog header file that defines all the constants required by the realtime API.

To use the Verilog realtime API you need to include this file into the Verilog prior to making use of any of the realtime API defined constants.

$cm_coverage

The task $cm_coverage takes the following arguments:

```
$cm_coverage(mode, type, scope, module_or_instance+)
```

and returns an integer value.

Arguments

The arguments to $cm_coverage are defined as follows:

mode. Integer-valued argument. Selects the function to be performed. It can be any of the following:

- `CM_START - Starts or restarts coverage for the given coverage type for the specified portion of the design. Has no effect if the supplied coverage type is already running. Equivalent to executing the appropriate $start command.
- `CM_STOP - Stops coverage for the given coverage for the given portion of the design. Equivalent to executing the appropriate $stop command.
- `CM_CHECK - Checks whether the given coverage is available for the specified portion of the design.

type. Integer-valued argument. Identifies the type of coverage information required. Can be one of the following:

- `CM_SOURCE - Line coverage.
- `CM_CONDITION - Condition coverage.
- `CM_TOGGLE - Toggle coverage.
- `CM_FSM - FSM coverage.
- `CM_FSM_TRANS - FSM coverage.
- `CM_FSM_STATE - FSM coverage.
- `CM_UEXPR - User expression coverage.

Note that for the purposes of $cm_coverage, ‘CM_FSM, ‘CM_FSM_TRANS, and ‘CM_FSM_STATE are considered identical. Specifically any and all of these act upon FSM coverage.

scope. Integer-valued argument. Specifies how the module_or_instance argument will be interpreted and can be either:

- `CM_MODULE - Modifies module_or_instance arguments to refer only to the specifically named modules or instances.
- `CM_HIER - Modifies module_or_instance arguments to refer to the given modules or instances and additionally to the hierarchy below said modules or instances.

module_or_instance. This is a set of one or more string-valued arguments that, together with the scope argument, identify the portions of the design for which coverage data is required. The interaction is as follows:

- `CM_MODULE + module_name - Action to be taken for all instances of the given module.
- `CM_MODULE + instance_name - Just for the specific instance named.
- `CM_HIER + module_name - All instances of the given module and additionally the entire design hierarchy underneath each such instance.
- `CM_HIER + instance_name - The specific instance named plus the entire design hierarchy underneath that instance.

Return Value

The return value from the $cm_coverage function depends on the mode. For each mode:

mode=‘CM_CHECK.

- `CM_NOERROR - The specified coverage type is available in this simulation for the given portion of the design hierarchy.
- `CM_ERROR - An invalid argument was supplied.
- `CM_NOCOV - The specified coverage type is not available in this simulation for the given portion of the design hierarchy.
- `CM_PARTIAL - The coverage type supplied is not fully available in this simulation for the given portion of the design hierarchy. Similar to `CM_NOCOV, but indicates that only a portion of the supplied hierarchy is missing this coverage, rather than this coverage being entirely missing from that portion of the design.

mode=‘CM_START.

- `CM_NOERROR - Specified coverage successfully started in the specified portion of the design.
- `CM_ERROR - An invalid argument was supplied.
- `CM_NOCOV - This coverage cannot be enabled for the given portion of the design. Generally implies that the design has not been instrumented for the given coverage type.
- `CM_PARTIAL - The specified coverage data is not fully available in this simulation. Similar to `CM_NOCOV, but indicates that this coverage is missing only from a portion of the supplied hierarchy.

mode=‘CM_STOP.

- `CM_NOERROR - The specified coverage was successfully stopped in the given portion of the design.
- `CM_ERROR - An invalid argument was supplied.

$cm_get_coverage

The task $cm_get_coverage takes the following arguments:

```
$cm_get_coverage(type, scope, module_or_instance+)
```

and returns an integer value.

Arguments

The arguments to $cm_get_coverage are defined as follows:

type. integer-valued argument. Identifies the type of coverage information required. It can be any of the following values:

- `CM_SOURCE - Line coverage. Coverage returned in terms of the number of blocks covered.
- `CM_CONDITION - Condition coverage. Coverage returned is the number of condition vectors covered.
- `CM_TOGGLE - Toggle coverage. Coverage returned is the number of net bits plus number of reg bits covered.
- `CM_FSM - FSM transition coverage (same as `CM_FSM_TRANS).
- `CM_FSM_TRANS - FSM transition coverage. Coverage returned is the number of legal transitions covered.

- `CM_FSM_STATE - FSM state coverage. Coverage returned is the number of legal states covered.
- `CM_UEXPR - User-expression coverage.

scope. Integer-valued argument. Defines the extent of the scope for which coverage information is requested. This extent will apply to all the following arguments. It can be one of the following values:

- `CM_MODULE - Modifies module_or_instance arguments to refer only to the specifically named modules or instances.
- `CM_HIER - Modifies module_or_instance arguments to refer to the given modules or instances and additionally the hierarchy below said modules or instances.

module_or_instance. String-valued arguments. This set of one or more arguments identifies the portions of the design for which coverage data is required. The type of this argument together with the scope argument determine the portion of the design from which coverage data will be obtained:

- CM_MODULE + module_name - Sum of coverage data for all instances of the given module.
- CM_MODULE + instance_name - Coverage data for the given instance.
- CM_HIER + module_name - Sum of coverage data for all instances of the given module including the entire hierarchy beneath each such instance.
- CM_HIER + instance_name - Coverage data for the given instance including the entire hierarchy beneath that instance.

Return Value

$cm_get_coverage returns one of the following values:

- 0 to MAXINT - A positive integer that represents the current coverage for the specified portion of the model. Note that the magnitude of this value has no special meaning other than relative to the maximum coverage obtained from $cm_get_limit.
- `CM_ERROR - An invalid argument was supplied.
- `CM_NOCOV - The specific type of coverage requested is not available for the given portion of the design. This might be due to a number of causes, such as this coverage not having been started or the design not having been instrumented for this type of coverage.

The actual coverage number description depends on the type of coverage, as follows:

- `CM_SOURCE - Coverage number indicates the number of basic blocks that have been covered
- `CM_CONDITION - Coverage number indicates the number of condition vectors observed.
- `CM_FSM_TRANS - Coverage number indicates the number of legal transitions observed.
- `CM_FSM_STATES - Coverage number indicates the number of legal states observed.
- `CM_TOGGLE - Coverage number returns the total number of net bits plus reg bits that have been observed to toggle.
- `CM_UEXPR - Coverage number indicates the number of user expression vectors observed.

It is important to note that a valid coverage number is returned even if the requested coverage is only partially available in the specified hierarchy. Whether this coverage is fully available in the specified hierarchy can be determined via the use of the $cm_coverage task.

$cm_get_limit

The task $cm_get_limit takes the following arguments:

```
$cm_get_limit(type, scope, module_or_instance+)
```

and returns an integer value.

Arguments

The arguments to $cm_get_limit are identical in definition to those for $cm_get_coverage.

Return Value

$cm_get_limit returns one of the following values:

- 0 to MAXINT - A positive integer which represents the maximum possible coverage value for the specified portion of the model. The coverage values returned by $cm_get_coverage can only be interpreted with relation to this limit, with a coverage value equal to the limit representing 100% coverage.
- `CM_ERROR - An invalid argument was supplied.
- `CM_NOCOV - The specific type of coverage requested is not available for the given portion of the design. This might be due to a number of causes, such as this coverage not having been started or the design not having been instrumented for this type of coverage.

The actual coverage number description depends on the type of coverage, as follows:

- `CM_SOURCE - Limit number indicates the total number of basic blocks.
- `CM_CONDITION - Limit number indicates the total number of condition vectors.
- `CM_FSM_TRANS - Limit number indicates the total number of legal transitions.
- `CM_FSM_STATES - Limit number indicates the total number of legal states.
- `CM_TOGGLE - Limit number returns the total number of net bits plus reg bits.
- `CM_UEXPR - Limit number indicates the total number of user expression vectors.

It is important to note that a valid limit number will be returned even if the requested coverage is only partially available in the specified hierarchy. Whether this coverage is fully available in the specified hierarchy can be determined via the use of the $cm_coverage task.

C Interface to Realtime Coverage

Header File

To use the C API to realtime coverage, users have to include the header file cm_realtime.h into their code. This header file contains the types and prototypes to the functions provided by the RealTime Coverage API.

Types and Constants

cm_status_t

This is an enum, and is used for the return values from most of the RTC API functions. The values in this enum are:

CM_NOERROR, CM_ERROR, CM_NOCOV, CM_PARTIAL

cm_mode_t

This is an enum and is used as the type of the mode arguments. The values in the enum are:

CM_START, CM_STOP, CM_CHECK

cm_cover_t

This is an enum and is the type of all the cover arguments. The values in the enum are:

CM_SOURCE, CM_CONDITION, CM_TOGGLE, CM_FSM, CM_FSM_TRANS, CM_FSM_STATE, CM_UEXPR

cm_scope_t

This is an enum and is the type of all the scope arguments. The values in the enum are:

CM_MODULE, CM_HIER

CM_VERSION

This is a constant and denotes the version of the API corresponding to this interface.

Functions

cm_api_version

```
int cm_api_version()
```

This function returns a 4-byte integer denoting the version of the RTC API implemented by the runtime library. The least-significant-bit encodes whether the RTC API is the 64-bit version of the library or the 32-bit version of the library.

If the version number is odd (least significant bit set) then the RTC API is being provided by a 64-bit runtime library. Otherwise it is being provided by the 32-bit runtime library.

The remaining bits encode the version of the API implemented by the library and must be compared to the version number given in the CM_VERSION constant.

Any mismatch between the version number or between 32/64-bit implementation between the library and the application imply that the application is using an incompatible version of the RTC API. If this is the case, none of the remaining API functions can be safely used.

Sample usage:

```
int version = cm_api_version();
#ifdef SUN64
if (!(version & 0x1)) {
    fprintf(stderr, "64-bit application linked to 32-bit
library. Exiting...\n");
    exit(1)
} else {
    version &= 0x1; /* strip out 64-bit marker */
}
#else
```

```
if (version & 0x1) {
    fprintf(stderr, "32-bit application linked to 64-bit
library. Exiting...\n");
    exit(1)
}
#endif
if (version != CM_VERSION) {
    fprintf(stderr, "Incompatible version detected.
Exiting...\n");
    exit(1);
}
```

cm_coverage

```
cm_status_t
cm_coverage(cm_mode_t, cm_cover_t, cm_scope_t, const char*
mod_or_inst)
```

Functionally has the same behavior as the $cm_coverage task. However, only one module or instance argument may be supplied. The return value is as per the $cm_coverage task.

The mod_or_inst argument must be a valid C string and must not be NULL. If these conditions are not met the results are unpredictable.

cm_get_coverage

```
cm_status_t
cm_get_coverage(int* coverage, cm_cover_t, cm_scope_t,
const char* mod_or_inst)
```

Functionally has the same behavior as the $cm_get_coverage task.

The coverage pointer must be a pointer to a valid address large enough to store an integer.

The mod_or_inst argument must be a valid C string and must not be NULL. If these conditions are not met the results are unpredictable.

The return code always describes the success or otherwise of the function. If the function returns CM_NOERROR, the *coverage contains the current coverage for the given type of coverage for the given portion of the design.

cm_get_limit

```
cm_status_t
cm_get_limit(int* limit, cm_cover_t, cm_scope_t, const char*
mod_or_inst)
```

Functionally has the same behavior as the $cm_get_limit task.

The limit pointer must be a pointer to a valid address large enough to store an integer.

The mod_or_inst argument must be a valid C string and must not be NULL. If these conditions are not met the results are unpredictable.

The return code always describes the success or otherwise of the function. If the function returns CM_NOERROR, the *coverage contains the current coverage for the given type of coverage for the given portion of the design.

cm_instance_id

```
int
cm_instance_id(const char* instance)
```

-1 if no such instance known, otherwise an integer representing the ID of that instance.

cm_count_fsms

```
int
cm_count_fsms(int instance_id)
```

Returns the number of FSMs known in the given instance.

instance_id must be a valid id as returned by cm_instance_id. Otherwise results are unpredictable.

cm_fsm_id

```
int
cm_fsm_id(int instance_id, int index)
```

Returns the fsm_id of the indexth FSM in the given instance. Returns -1 if there are no FSMs in the given instance or if an invalid index was supplied.

instance_id must be a valid id as returned by cm_instance_id(). Otherwise results are unpredictable.

The valid range for index is $0 \leq index < cm\_count\_fsms(instance\_id)$. Any indexes outside this range cause the function to return -1.

cm_fsm_get_statevar

```
char*
cm_fsm_get_statevar(int instance_id, int fsm_id)
```

Returns the name of the state variable for the given FSM. Can return NULL for implied state FSMs or if an error is detected.

instance_id must be a valid id as returned by cm_instance_id(). Otherwise results are unpredictable.

fsm_id must be a valid id as returned by cm_fsm_id(). Otherwise results are unpredictable.

cm_fsm_count_states

```
int
cm_fsm_count_states(int instance_id, int fsm_id)
```

Returns the number of states in the given FSM. Returns -1 if an error is detected.

instance_id must be a valid id as returned by cm_instance_id(). Otherwise results are unpredictable.

fsm_id must be a valid id as returned by cm_fsm_id(). Otherwise results are unpredictable.

cm_fsm_get_state

```
cm_status_t
cm_fsm_get_state(long long* state, int* covered,
int instance_id, int fsm_id, int index)
```

Use to obtain the value corresponding to the indexth state in the given FSM and the coverage of that specific state. State values are at maximum 64-bit values and are always plain 2-state binary (no Xs or Zs) and can therefore be adequately represented by C 64-bit integers (long long).

The return value is CM_NOERROR if the operation succeeds, CM_ERROR otherwise.

instance_id must be a valid id as returned by cm_instance_id(). Otherwise results are unpredictable.

fsm_id must be a valid id as returned by cm_fsm_id(). Otherwise results are unpredictable.

The valid range for index is 0 ≤ index < cm_fsm_count_states(...). Outside this range the function always returns CM_ERROR.

cm_fsm_count_transitions

```
int
cm_fsm_count_transitions(int instance_id, int fsm_id)
```

Returns the number of transitions in the given FSM. Returns -1 if an error is detected.

instance_id must be a valid id as returned by cm_instance_id(). Otherwise results are unpredictable.

fsm_id must be a valid id as returned by cm_fsm_id(). Otherwise results are unpredictable.

cm_fsm_get_transition

```
cm_status_t
cm_fsm_get_transition(long long* from, long long* to, int*
covered, int instance_id, int fsm_id, int index)
```

Used to obtain information about a specific transition in the given FSM, specifically from what state to what state the transition occurs and whether this transition has been covered. The state representation is the same as used by cm_fsm_get_state.

The return value is CM_NOERROR if the operation succeeds, CM_ERROR otherwise.

instance_id must be a valid id as returned by cm_instance_id(). Otherwise results are unpredictable.

fsm_id must be a valid id as returned by cm_fsm_id(). Otherwise results are unpredictable.

The valid range for index is 0 ≤ index < cm_fsm_count_transitions(...). Outside this range the function always returns CM_ERROR.

Linking

The realtime API is present in the cmMonitor library. These functions are present only in this library and thus can only be used if it is linked into the simulator.

To break this dependency when the realtime API is not required the client application has to supply a set of dummy functions to be linked in only when the realtime API is not required.