User Manual

FSM Generator

(v1.1)

Summary

The purpose of this user manual is to describe the method to use the FSM code generation tool

Related Document:

Table of Contents

Index of Tables

Index of Figures

1. Summary

The purpose of this document is to describe the method to use the Finite-state machine (FSM) code generation tool. This document also describes the format of input files and shows sample output files of the script.

2. FSM generator

2.1. Overview

Currently, the design method is different from designers. That difference leads to difficulties in design itself due to the lack of a standard procedure. Moreover, it is also difficult for the verifier in checking the design and for the reviewer in both reviewing source code and confirming number of test cases. A standard design method should be applied solve this difficulty. Finite-state machine, which is a suitable model of computation for both computer programs and sequential logic circuits, will be applied in the development flow.

FSM code generation tool (FSM generator) is a script which generates the source code of FSM control part of a model. It is written in Python.

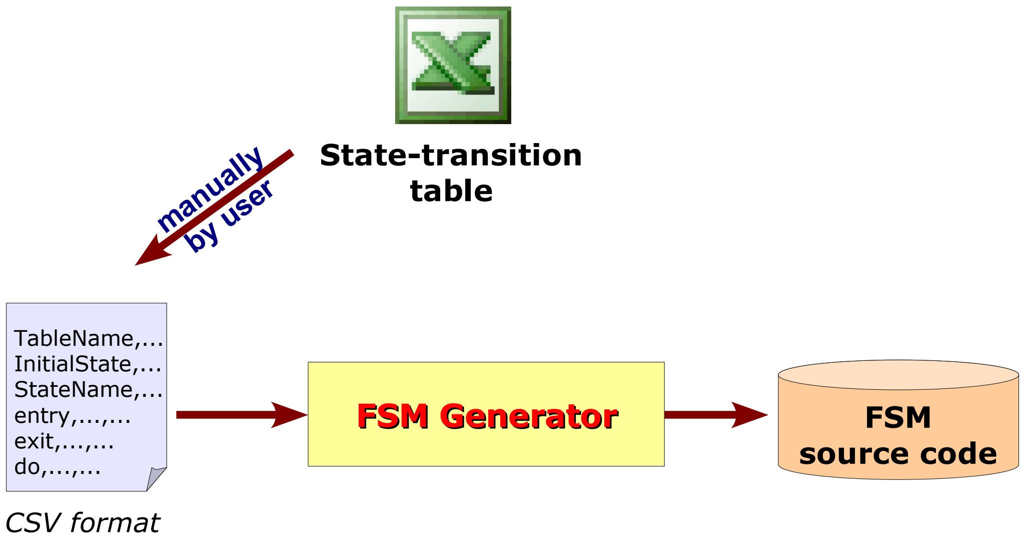

Figure 2.1 FSM Generator application

Explanation:

(1) FSM Generator reads an input text file which contains model's state-transition table, then generates SystemC source code for FSM control part. The input text file stores state-transition information in table structure (Comma Separated Value - CSV format). Therefore, this file can be generated from a Spreadsheet file (for example, file made by MS Excel or OpenOffice scalc). This step can be done manually by user.

(2) FSM control part can be used directly without any edition.

2.2. Feature list

Table 2.1 List of supported features of Generator

Feature	Description
Python script	Support Python 3.1.2 or higher
CSV input files	- Support CSV format input. - Support multiple input files.
Flat-type and Hierarchical-type of input state-transition table (STT)	- Single FSM class output for flat-type STT - Multiple FSM class output for hierarchical-type STT
Output language	C++
Class name and file name	- Name of STT specifies output class name and file name, with “_fsm” suffix. - Class name starts with “C” prefix. - File name does not start with “C” prefix.
Complex condition	Support format for complex if-then-else condition branches in STT.

Table 2.2 List of supported features of FSM class

Feature	Description
Determine current state	A member data to specify the current state.
Control state-transition	- Receive the trigger from model by function-call. - Calculate the next state based on the current state and the event triggered.
Proceed state-transition	- Update the current state. - Call model's action functions to perform exit and entry actions. - Proceed next state-transition if the next state is an non-triggered state.
Debug message	- Support an API to enable/disable dumping messages. - Dump message for all state-transitions.
Support C++ template	Generate template-related code if model uses template.

3. Usage

3.1. Files

Table 3.1 lists all necessary files for running FSM Generator.

Table 3.1 File list

File name	Revision	File description
fsmgen.py	**v2014_07_03**	The main script
fsm_h.skl		Skeleton file for .h file generation
fsm_cpp.skl		Skeleton file for .cpp file generation
fsmif_h.skl		Skeleton file for FSM IF file generation
copyright.txt		Copyright content

3.2. Syntax

Table 3.2 The usage of FSM Generator.

Type	Content
Script	fsmgen.py
Language	Python (version 3.1.2 or higher)
Purpose	This script is used to generate FSM class based on state-transition table file.
Usage	python fsmgen.py **[option] --name *name* <input_files>**
Option	**"--version"** (or **-v**): print script version information – Default value: none **"--help"** (or **-h**): print script usage information – Default value: none
Argument	***name*** : specify the model class name.
Input file	State-transition table file(s) in CSV format
Output file	FSM class files, file name are specified by state-transition table name.

3.3. Input

The only input file of the generator is the FSM table file(s). This is a CSV-format file which contains state-transition table (STT) of the model. Contents in this file are separated by commas (,).

In a model, there may be more than one state-transition tables. Multiple STT is realized in the hierarchical state machine, which contains hierarchically nested states. In other words, a parent FSM may include one or more sub FSM, which can be put in one or more FSM table files.

3.3.1. State-transition table contents

Table 3.3 describes the content of all possible fields in state-transition tables.

Table 3.3 Contents of a state-transition table

Keyword	Mandatory/ Optional	Meaning	Description
TableName	M	STT name	The name of STT.

			It defines the name of generated FSM class.
InitialState	M	Initial state	The initial state of the FSM. This is the initial value of CurrentState variable of FSM class.
StateName	M	List of states	All states the FSM has. This is used as the title for all below entries.
entry	M	Entry actions	The function which is called upon entry to the state.
exit	M	Exit actions	The function which is called upon exit from the state.
do	M	Do actions	The function which is called when staying in the state.
include	M	Sub FSM table	Define the sub-FSM which is invoked upon entry to the state.
EventName	O	State-transition triggered by the event	Event that affects the FSM. The state-transition which is triggered when event occurs is defined for each state. The next two entries (Start/End) is special (optional) event which is reserved for Sub FSM only
Start	O	Start event of the Sub FSM	Event to start the Sub FSM. It is triggered by the parent FSM.
End	O	End event of the Sub FSM	Event to finish the Sub FSM. It is triggered by the parent FSM.
WOE	O	Without event	Another state-transition happens continuously after entry to a state.
template	O	Model class uses template	- Indicate the template definition in model class, which should be generated same in FSM class. - “template” information should follow the “TableName” information, on the same line. - Format: **template** <argument: type, ...> : <value1> <value2> … - Example: **template** <BUSWIDTH:unsigned int, N:char>: <16,1> <16,2> <32,1> <32,2>
[and **]**	O	Guard condition	- Specify the condition to select the next-state or action function (entry/exit/do). - Guard condition of next-state should be exclusive. - Guard condition of action function can be specified in complex combination. The Figure 3.1 describes the supported format for guard condition of action function.

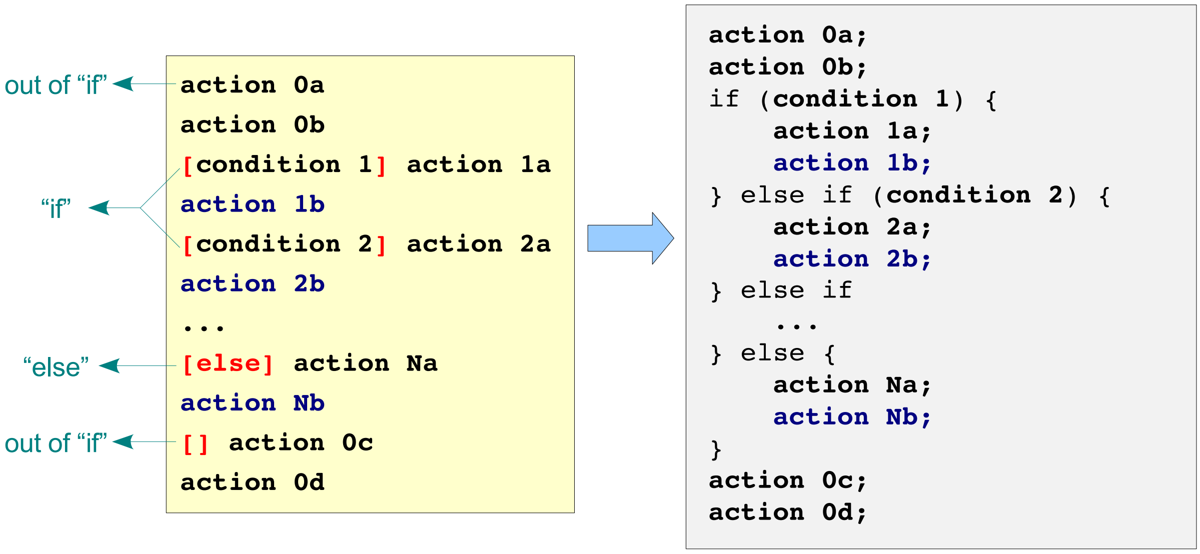

Figure 3.1 Format of guard condition.

3.3.2. Example

This section demonstrates an example of STT taking a simple counter as the sample model. Table 3.4 describes briefly the specification of the simple counter model, including list of events, conditions and corresponding behavior of the model.

Table 3.4 Simple counter model specification

Event	Condition	Operation
Start counter	-	Counter starts/continues counting up from current value.
Stop counter	-	Counter stops counting.
Counter is overflow (maximum value 0xFFFFFFFF)	Interrupt enabled	Issue an interrupt.
	Interrupt disabled	Not issue an interrupt.
Interrupt issue process is done (either enabled or disabled)	One-shot mode	Counter stops counting and its value is cleared to 0.
	Free-run mode	Counter continues counting up from 0.
Input capture	Counter is counting	Current counter value is displayed.
Reset	-	Model is reset

Figure 3.2 shows the STT of the simple counter. This STT is created in hierarchical-type.

TableName	Counter_main	
InitialState	ACTIVE	
StateName	RESET	ACTIVE
entry	EnableReset(true)	-
exit	EnableReset(false)	Counter_core->End()
do	-	-
include	-	Counter_core
ResetOn	/	RESET
ResetOff	ACTIVE	/

TableName	Counter_core			
InitialState	IDLE			
StateName	IDLE	COUNT	INTERRUPT	CAPTURE
entry	-	SetSTR(true)	IssueInterrupt()	InputCapture()
exit	-	SetSTR(false)	-	-
do	-	-	-	-
include	-	-	-	-
StartCount	COUNT	/	/	/
StopCount	/	IDLE	/	/
End	/	IDLE	/	/
Capture	/	CAPTURE	/	X
Overflow	X	[(parent->int_en == 0)&&(parent->mode == 0)] IDLE [(parent->int_en == 0)&&(parent->mode == 1)] COUNT [parent->int_en == 1] INTERRUPT	X	/
WOE	X	X	[parent->mode == 0] IDLE [parent->mode == 1] COUNT	COUNT

Legends:

	State name	-	Not exist
	Actions	/	Ignore the event in the state
	Sub-FSM	**X**	Prohibit the event in the state
	Events	[ ]	Guard condition
	State-transition		

Figure 3.2 Example of an input STT.

Figure 3.3 shows the corresponding CSV-format file.

```
"TableName","counter","template <N:unsigned int,P:char>: <1,2> <1,3> <2,3>",,
"InitialState","ACTIVE",,,
"StateName","RESET","ACTIVE",,
"entry","EnableReset(true)","-",,
"exit","EnableReset(false)","counter_core->End()",,
"do","-","-",,
"include","-","counter_core",,
"ResetOn","/","RESET",,
"ResetOff","ACTIVE","/",,
,,,,
,,,,
"TableName","counter_core",,,
"InitialState","IDLE",,,
"StateName","IDLE","COUNT","INTERRUPT","CAPTURE"
"entry","mParent->EntryIdleSub()","mParent->SetSTR(true)","mParent-
>IssueInterrupt()","mParent->InputCapture()"
"exit","-","mParent->SetSTR(false)","-","-"
"do","-","-","-","-"
"include","-","-","-","-"
"Start","COUNT","/","/","/"
"End","/","IDLE","/","/"
"Capture","/","CAPTURE","/","/"
"Overflow","X","[(parent->int_en == 0)&&(parent->mode == 0)] IDLE
[(parent->int_en == 0)&&(parent->mode == 1)] COUNT
[parent->int_en == 1] INTERRUPT","X","/"
"WOE","X","X","[parent->mode == 0] IDLE
[parent->mode == 1] COUNT","COUNT"
```

new line
in cell

Figure 3.3 Example of CSV-format file.

3.4. Output

3.4.1. Output file

Table 3.5 List of output files

Output file name	Class name	Description
***top-level-STTname*_fsmif.h** (one file only)	-	FSM IF code lines to be included in model class
***top-level-STT-name*_fsm.h** (one file only)	Cfsm_base C*STTname*_fsm C*SubSTTname*_fsm	Implementation of FSM base class Declaration of top-level FSM class Declaration of sub-level FSM class(es)
***STTname*_fsm.cpp** (one file for each STT)	C*STTname*_fsm C*SubSTTname*_fsm	Implementation of FSM class(es)

Note:

(*) *STTname* is specified by the TableName in the STT. *top-level-STTname* is the TableName of the top-level STT.

(*) Sub-level FSM class is generated only

3.4.2. Class relationship

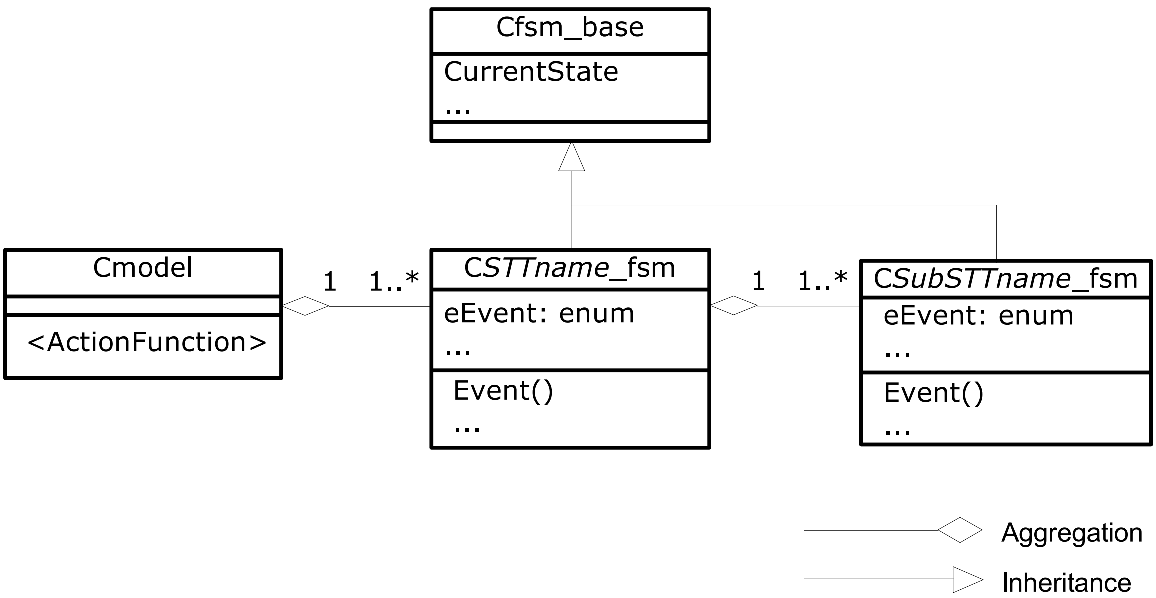

Figure 3.4 FSM classes and model class relationship.

Explanation:

(1) Model class instantiates C*STTname*_fsm class (so-called top-level FSM class).

(2) Model class calls Event() function of FSM class to trigger an event when it occurs. FSM class controls the model's state-transition and calls the functions of model class for entry/exit/do actions (using *parent* pointer).

(3) In hierarchical design, top-level FSM class may instantiate one or more sub-level FSM class (*CSubSTTname*_fsm). Model class can trigger an event of a sub-level FSM class directly via top-level FSM class.

3.4.3. Function-call structure

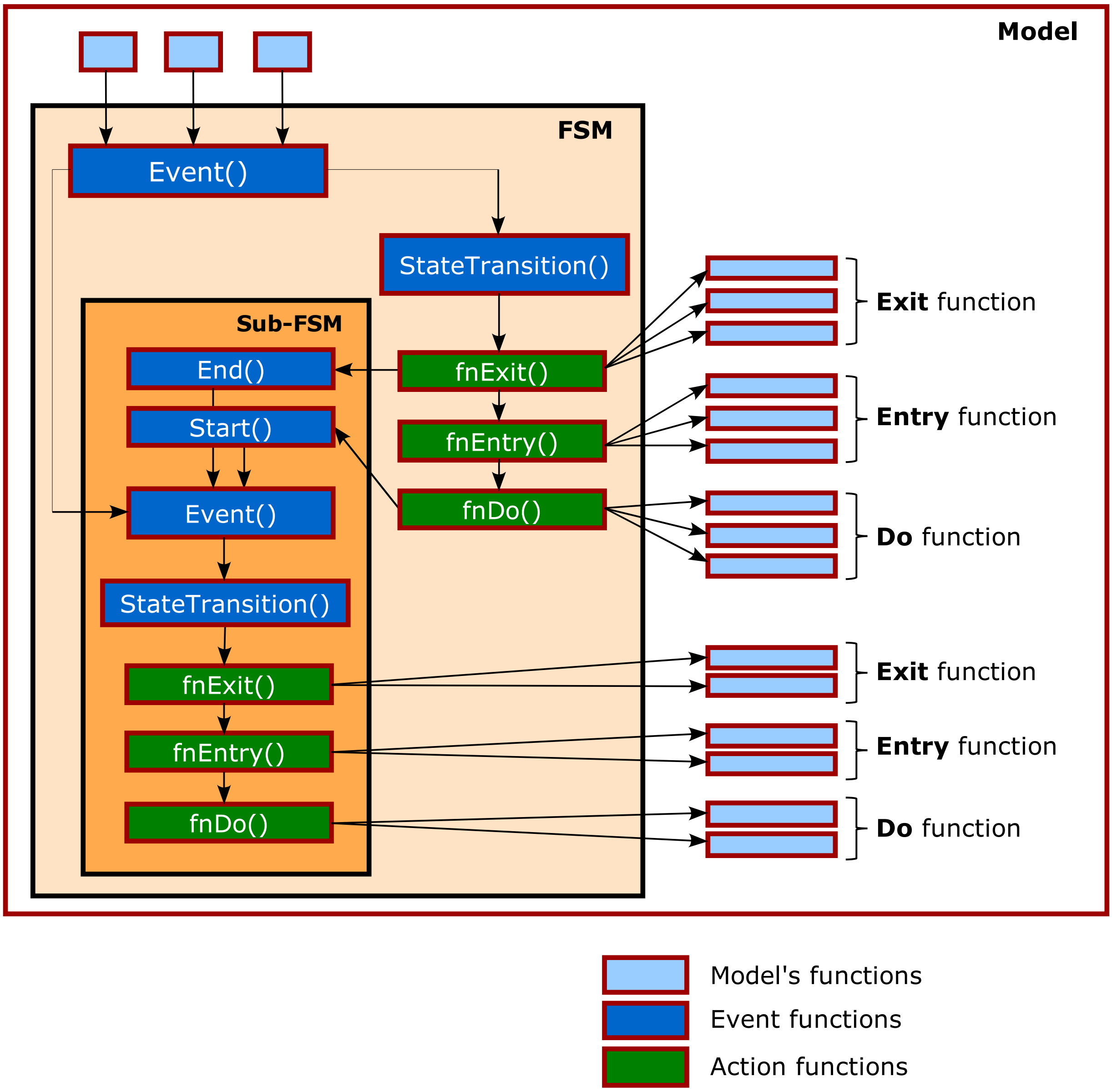

Figure 3.5 Function-call structure between FSM class and model class.

Explanation:

- Model calls Event() of top-level FSM class to trigger an event to change the state of FSM.
- Event() function checks the triggered event belongs to its STT or to one of its sub-level STT.
  - If the event belongs to itself: call StateTransition().
  - If the event is from one of sub-level class: calls the Event() of the corresponding class to pass the trigger. The sub-level Event() will call its own StateTransition() for state transition.
- StateTransition() function:
  - Calculate the next state based on the current state. If the event is forbidden in a state, an assertion error will be raised (by sc_assert).
  - Process the state-transition by calling functions in below order:

    - Call fnExit() -> fnEntry() -> fnDo()
    - Check if the next state is non-triggered state ("WOE" of next state exists or not). If yes, the transition to next state continue automatically until being stopped at a triggered state.

- Start() and End() are special functions in the Sub-FSM which are used to trigger the Start and End events of the Sub-FSM respectively. They are called by the parent FSM directly.
- fnExit(), fnEntry() and fnDo(): call model's functions to proceed actions on state-transition.

3.4.4. API functions

Table 3.6 List of API functions of FSM class

No.	Function syntax	Description
1	void Event (eEvent event)	Trigger an event based on input argument
2	unsigned int GetCurrentState (void)	Return the current state
3	void EnableDumpStateTrans (bool enable)	Enable dumping information of state-transition.

3.4.5. Sample code

Figures 3.6 to 3.9 demonstrate the source code in output files, also taking the simple counter as example.

➢ **Header (.h) file**

FSM base class

```
#include "systemc.h"
#include <string>

class Ccounter;
class Ccounter_core_fsm;

/// FSM base class
class Cfsm_base
{
public:
    Cfsm_base(Ccounter *_parent, std::string upper_state, unsigned int
num_of_state, unsigned int num_of_event)
    {
        mNumOfEvent = num_of_event;
        mNumOfState = num_of_state;
        ...
    }
    ...
    virtual void Event(unsigned int) = 0;
    unsigned int GetCurrentState(void)
    {
        return mCurrentState;
    }
    virtual void EnableDumpStateTrans(bool enable) = 0;
protected:
    Ccounter *mParent;
    unsigned int mNumOfEvent;
    unsigned int mNumOfState;
    unsigned int mCurrentState;
    unsigned int mPreState;
    unsigned int mNextState;
    ....
    bool StateTransition(unsigned int event)
    {
        bool trans_next = false;
        unsigned int next_state = mNextStateList[mCurrentState][event];
        if (next_state > mNumOfState) { // next state depends on condition
            CheckCondition(next_state - mNumOfState - 1);
        } else {
            mNextState = next_state;
        }
        if ((mNextState != mNumOfState) && (mNextState != mCurrentState)) {
            DumpStateTransInfo();
            fnExit();
            fnEntry();
            fnDo();
            if (mNextStateList[mCurrentState][mNumOfEvent-1] != mNumOfState) {
                trans_next = true;
            }
        }

        return trans_next;
    }
};
....
```

Figure 3.6 Sample .h file (1/2).

```
class Ccounter_fsm: public Cfsm_base
{
friend class Ccounter;
public:
    enum eState {
        emStRESET
       ,emStACTIVE
       ,emStNA
    };
    enum eEvent {
        emEvtResetOn
       ,emEvtResetOff
       ,emEvtWOE
       // Ccounter_core_fsm
       ,emEvtStartCount
       ,emEvtStopCount
       ,emEvtCapture
       ,emEvtOverflow
       // End of event list
       ,emTotalNumOfEvent
    };
    struct SEventFunctionCallInfo {
        Cfsm_base *pFSMObject;    // Sub-FSM pointer
        unsigned int event_index; // Event index in the Sub-FSM
    };

    Ccounter_fsm(Ccounter *_parent, std::string upper_state = "");
    ~Ccounter_fsm(void);
    void Event(unsigned int event);
    void EnableDumpStateTrans(bool enable);
private:
    Ccounter_core_fsm* pCcounter_core_fsm;
    SEventFunctionCallInfo mEventFuncTable[emTotalNumOfEvent];
    void CheckCondition(const unsigned int condition_id);
    void fnEntry(void);
    void fnDo(void);
    void fnExit(void);
    void DumpStateTransInfo(void);
};
```

top-level FSM class

State enumeration

Event enumeration

Instantiation of sub-level FSM class

```
class Ccounter_core_fsm: public Cfsm_base
{
friend class Ccounter;
friend class Ccounter_fsm;
public:
    enum eState {
        emStIDLE
       ,emStCOUNT
       ,emStINTERRUPT
       ,emStCAPTURE
       ,emStNA
    };
    enum eEvent {
        emEvtStartCount
       ,emEvtStopCount
       ,emEvtCapture
       ,emEvtOverflow
       ,emEvtWOE
    };
    ....
};
....
```

sub-level FSM class

Figure 3.7 Sample .h file (2/2).

➢ Implementation (.cpp) file

```
#include "counter_fsm.h"
#include "counter.h"

Ccounter_fsm::Ccounter_fsm(Ccounter *_parent, std::string upper_state)
    :Cfsm_base(_parent, upper_state, emStNA, emEvtWOE+1)
{
    mStateNameStr[emStACTIVE] = "ACTIVE";
    mStateNameStr[emStRESET ] = "RESET";

    mCurrentState =  emStACTIVE; // Initial state

    pCcounter_core_fsm = new Ccounter_core_fsm(parent);

    // State transition table construction
    mNextStateList[emStRESET    ][emEvtResetOff  ] = emStACTIVE;

    mNextStateList[emStACTIVE   ][emEvtResetOn   ] = emStRESET;

    // Event function pointer table construction
    SEventFunctionCallInfo event_table_tmp[emTotalNumOfEvent] = {
        { this , emEvtResetOn},
        { this , emEvtResetOff},
        { this , emEvtWOE},
        // Ccounter_core_fsm
        {pCcounter_core_fsm, (unsigned int) Ccounter_core_fsm::emEvtStartCount},
        {pCcounter_core_fsm, (unsigned int) Ccounter_core_fsm::emEvtStopCount},
        {pCcounter_core_fsm, (unsigned int) Ccounter_core_fsm::emEvtCapture},
        {pCcounter_core_fsm, (unsigned int) Ccounter_core_fsm::emEvtOverflow}
    };
    for (unsigned int i=0; i<emTotalNumOfEvent; i++) {
        mEventFuncTable[i] = event_table_tmp[i];
    }

}

Ccounter_fsm::~Ccounter_fsm(void)
{
    delete pCcounter_core_fsm;
}

void Ccounter_fsm::Event(eEvent event)
{
    sc_assert(event != emEvtWOE);

    if (mEventFuncTable[event].pFSMObject == this) {
        bool state_next = StateTransition(event);
        while (state_next) state_next = StateTransition(emEvtWOE);
    } else {
        // call Event() of sub-FSM
        ((mEventFuncTable[event].pFSMObject)->*pEventFunc)
(mEventFuncTable[event].event_index);
    }
}

void Ccounter_fsm::CheckCondition(const unsigned int condition_id)
{
}
....
```

initial state

Sub-FSM allocation

Next-state table

Pointer to Event() of sub-level FSM classes (generated in top-level class only)

calls Event() of sub-level FSM (generated in top-level class only)

Figure 3.8 Sample .cpp file (top-level FSM class)

```
#include "counter_fsm.h"
#include "counter.h"

Ccounter_core_fsm::Ccounter_core_fsm(Ccounter *_parent, std::string upper_state)
    :Cfsm_base(_parent, upper_state, emStNA, emEvtWOE+1)
{
    mStateNameStr[emStIDLE     ] = mStateNamePrefix + "IDLE";
    mStateNameStr[emStCOUNT    ] = mStateNamePrefix + "COUNT";
    mStateNameStr[emStINTERRUPT] = mStateNamePrefix + "INTERRUPT";
    mStateNameStr[emStCAPTURE  ] = mStateNamePrefix + "CAPTURE";
```

initial state

```
    mCurrentState = emStIDLE; // Initial state
```

Next-state table

```
    // State transition table construction
    mNextStateList[emStIDLE     ][emEvtStartCount] = emStCOUNT;

    mNextStateList[emStCOUNT    ][emEvtStopCount ] = emStIDLE;
    mNextStateList[emStCOUNT    ][emEvtCapture   ] = emStCAPTURE;
    mNextStateList[emStCOUNT    ][emEvtOverflow  ] = emStNA + 1;

    mNextStateList[emStINTERRUPT][emEvtWOE       ] = emStNA + 2;

    mNextStateList[emStCAPTURE  ][emEvtWOE       ] = emStCOUNT;
}

Ccounter_fsm::~Ccounter_fsm(void)
{
    delete pCcounter_core_fsm;
}

void Ccounter_core_fsm::Event(unsigned int event)
{
    sc_assert(event != emEvtWOE);

    if (event == emEvtOverflow) {
        sc_assert(mCurrentState != emStIDLE);
    }

    bool state_next = StateTransition(event);
    while (state_next) state_next = StateTransition(emEvtWOE);
}

void Ccounter_core_fsm::CheckCondition(const unsigned int condition_id)
{
    sc_assert(condition_id < 2);
    switch (condition_id) {
```

Next-state table

```
        case 0:
            if ((mParent->int_en == 0)&&(mParent->mode == 0)) {
                mNextState = emStIDLE;
            }
            else if ((mParent->int_en == 0)&&(mParent->mode == 1)) {
                mNextState = emStCOUNT;
            }
            else if (mParent->int_en == 1) {
                mNextState = emStINTERRUPT;
            }
            break;
        case 1:
        ....
}
....
```

Figure 3.9 Sample .cpp file (sub-level FSM classes)

- **FSM IF (.h) file**

FSM IF file contains the source code for instantiation of FSM class in the model class. This file can be included into the content of model class. Figure 4.2 shows how to use the FSM IF file in model source code.

```
friend class Ccounter_fsm;

private:
    Ccounter_fsm* pCcounter_fsm;
    sc_event mCOUNTERFSMEvent[Ccounter_fsm::emTotalNumOfEvent];

    void COUNTERFSMInit(void)
    {
        pCcounter_fsm = new Ccounter_fsm(this);
        for (unsigned int i = 0; i < Ccounter_fsm::emTotalNumOfEvent; ++i) {
            sc_core::sc_spawn_options opt;
            opt.spawn_method();
            opt.set_sensitivity(&mCOUNTERFSMEvent[i]);
            opt.dont_initialize();
            sc_core::sc_spawn(sc_bind(&Ccounter::COUNTERFSMTriggerMethod, this,
i), sc_core::sc_gen_unique_name("CO
        }
    }

    void COUNTERFSMTriggerMethod(unsigned int event_code)
    {
        sc_assert(event_code < Ccounter_fsm::emTotalNumOfEvent);
        pCcounter_fsm->Event(event_code);
    }
```

SC_METHOD
for calling Event()

Figure 3.10 FSM I/F file.

4. Apply FSM Generator in model design

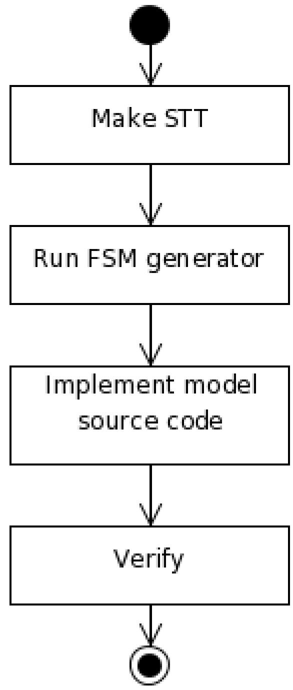

Figure 4.1 Model design with FSM Generator.

Table 4.1 Explanation of each steps of model design with FSM Generator.

No.	***Content***	***Description***
1	Make STT	Create the state-transition table of based on the model specification.
2	Run FSM generator	Run the FSM generator to generate the FSM files.
3	Implement model source code	Implement the source code of model to work with FSM class: • There is no affection on Register IF and Command IF. • Call Event() function to trigger events. • Implement action functions. Figure 4.2 below demonstrates the source code of the simple counter that cooperate with the FSM class shown in previous sections.
4	Verify	The verification of model source code is done as normal. With the FSM design, test items should at least cover all cases of state-transitions.

```
class Ccounter: public sc_module
{
#include "counter_fsmif.h"
public:
    SC_HAS_PROCESS(Ccounter);
    Ccounter(sc_module_name name) {
        COUNTERFSMInit();
        mode   = false;
        int_en = false;
        ....
    }

    void EnableReset(bool active)
    {
        if (active) {
            pCcounter_fsm->Event(pCcounter_fsm->emEvtResetOn);
        } else {
            pCcounter_fsm->Event(pCcounter_fsm->emEvtResetOff);
        }
    }

    void SetCounter(bool start)
    {
        if (start) {
            pCcounter_fsm->Event(pCcounter_fsm->emEvtStartCount);
        } else {
            pCcounter_fsm->Event(pCcounter_fsm->emEvtStopCount);
        }
    }

    void CaptureCounter(void)
    {
        pCcounter_fsm->Event(pCcounter_fsm->emEvtCapture);
    }

private:
    bool int_en; // enable interrupt
    bool mode;   // free-run or one-shot

    void SetSTR(bool on)
    {
        Cgeneral_timer::setSTR(on);
    }

    void IssueInterrupt(void)
    {
        interr.write(0);
    }

    void InputCapture(void)
    {
        mCaptureValue = Cgeneral_timer::getCNT();
    }
....
```

instantiate FSM class using FSMIF file

Model APIs triggers event of FSM class

Action functions

Figure 4.2 Implementation of sample model (simple counter)

Revision History				
Rev.	**Modified Contents**	**Approval**	**Reviewed by**	**Created by**
1.0	New creation in FSM generator (13032)	A.Imoto 04/15/2014	Vu Pham 04/03/2014	Duc Duong 04/03/2014
1.1	Updated in phase 2 (renamed to 14004): - Section 2.2: Add new feature to Table 2.1 and 2.2. - Section 3.1: Add file fsmif_h.skl to Table 3.1. - Table 3.3: Add format for "template" and "guard condition" - Add Figure 3.1 for the format of "guard condition" - Section 3.4.1: Update list of output file. - Section 3.4.2: Update class relationship. - Section 3.4.3: Update function-call structure. - Section 3.4.5: Update sample of generated files. - Figure 4.2: Update the source code of the sample model.		Vu Pham 07/07/2014	Duc Duong 07/07/2014