

A Tool for Converting Finite State Machines to SystemC

Tareq Hasan Khan, Ali Habibi[§], Sofiène Tahar, Otmane Ait Mohamed

Dept. of Electrical & Computer Engineering, Concordia University
Montreal, Quebec, Canada
Email: {tare_kha, tahar, ait}@ece.concordia.ca

[§]MIPS Technologies
Mountain View, California, USA
Email: habibi@mips.com

Technical Report

January 2007

Abstract

Finite state machines (FSM) are a basic component in hardware design, they represent the transformation between inputs and outputs for sequential designs. An FSM can be represented graphically, which would help the designer to visualize and design in a more efficient way, on the other hand the designer requires a fast direct way to convert the visualized design to hardware description languages (HDL) code directly in order to simulate and implement it for synthesis and analysis. In this report, we present a tool which starting from a graphical FSM representation produces SystemC code which can be used for analysis and simulation

Contents

1 Introduction	3
2 SystemC	4
3 FSM Representation in ASF Format	5
4 Tool Structure	6
5 Conclusion	11
Reference	12
Appendix A	
Tool User Interface and Source Code	14

1 Introduction

The behavior of computer systems can be described and analyzed by means of transition systems. Understanding and gaining more insight by inspecting these systems can be of great advantage when constructing complicated systems. The most commonly used transition systems are based on explicit state enumeration known as finite state machines (FSM). FSMs are a basic component of hardware designs, they represent the transformation between inputs and outputs for sequential designs. FSMs can be represented graphically, which would help the designer to visualize and design in a more efficient way. The designer requires a fast direct way to convert the visualized design to hardware description languages (HDL) code directly in order to simulate and implement it. CAD tools support built-in visualization software or interfaces to third party software. In the work presenting in the report, we use Active HDL State Editor [2] to draw and visualize the FSM. Active HDL [14] outputs a textual representation of the FSM in ASF (Active-HDL State machines Format) [2]. The proposed tool takes as input an ASF file and generates SystemC [4] code which can be used for simulation and analysis in SystemC environment.

Graphs are frequently used in computer applications as a general data structure to represent objects and relationships between them. They are used to implement hierarchies, dependency structures, networks, configurations, data flows, etc. Usually graph visualization tools support the following options: directed, undirected, and mixed graphs, hyper graphs, hierarchical graphs and graphical representations [5]. Different formats have been proposed as input to visualization tools. They usually consist of a language core to describe the structural properties of a graph and a flexible extension mechanism to add application-specific data. In this report, we use the new and rich ASF file format to represent FSMs. The Active HDL tool uses the ASF format to store graphical information to textual form and vice-versa. It also generates VHDL and Verilog code from the FSM. Similar visualization tools include the daVinci graph visualization [6] program and the VCG tool [7] which automatically computes the most optimal way to view the finite-state automaton by minimizing the number of crossing edges. Another visualization tool is AiSee [8], which is a part of the Absint static analyzer tool suite [8] and was developed initially to visualize the internal data structures found in compilers. Today it is widely used in many different areas including visualizing FSMs. AiSee automatically calculates a customizable layout of graphs specified in GDL (graph description language) [6]. This layout is then displayed, and can be printed or interactively explored. The Xilinx company provides a commercial tool for the rapid prototyping of a FSM design directly from the state diagram. Xilinx ISE tools [9] include an editor, named StateCAD, which allows users to graphically input state diagrams and translated them into a Verilog [11] behavioral HDL model. Finally in [20], the authors implemented a tool that takes dot FSM format and generates VHDL code. The tool presented in this report takes FSM description in ASF format and generates SystemC code. Tools that generate VHDL and Verilog code from graphical FSM is available, but generating SystemC code from graphical description is missing. This work fulfills this gap.

In the next section, we will discuss about the SystemC language. In Section 3, we will briefly overview the ASF Format. In Section 4, we will describe the proposed tool structure, SystemC Code generation algorithm and give an example of ASF to SystemC conversion. Finally, Section 5 concludes the paper.

2 SystemC

Modern hardware designers typically use HDLs to describe designs at various levels of abstraction. A hardware description language is a high level programming language, with the usual programming constructs such as assignments, conditions and iterations, as well as extensions for timing specification, concurrency and data structure suitable for modeling different aspects of hardware. The most popular hardware description languages are VHDL, Verilog and SystemC. We have chosen SystemC in this project as our target FSM code generation language.

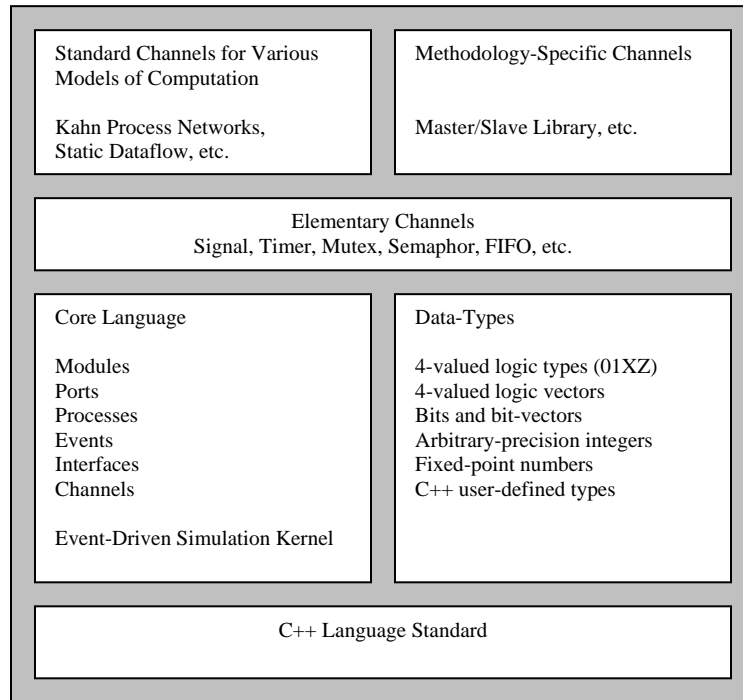


Fig 2.1 SystemC Language Structure

SystemC [4], one of the proposals of the electronic design automation (EDA) community has been set for the IEEE standard (IEEE1666-2005) library for system level design [12]. SystemC aims at bridging the gap between hardware and software design flows. Furthermore, it promotes the integration of different levels of abstraction in a unique design process. SystemC provides hardware-oriented constructs within the context of C++ as a class library implemented in standard C++. Fig 2.1 shows the various layers of SystemC. The bottom most layer highlights the fact that SystemC is built entirely on

standard C++. Its use spans design and verification from concept to implementation in hardware and software. SystemC provides an interoperable modeling platform which enables the development and exchange of very fast system-level C++ models. It also provides a stable platform for development of system-level tools.

3 FSM Representation in ASF Format

The ASF format is a new and reach file format to represent FSM in textual from. We used Active HDL State Diagram Editor to draw the FSM which outputs a textual representation of the FSM in ASF file format. The state editor has an attractive and easy to use user interface. It provides all the features to draw FSMs including some special features like setting Default State, Trap State, Transition Line Priority, Hierarchical FSM, etc [13]. The objects of an FSM like State, Label, Action, Transition Line, Condition and their properties are saved in the ASF file. In this project, we have used Active HDL 6.3 Student Edition which generates ASF file with version no 1.19. It is recommended to use this version of the tool. A portion of the format is shown at Fig 3.1

```

State:
    S [ID] [isDefStat/isTrapState]
Label:
    L [ID] [ObjectID] ... [Label Description]....[Label]
Action:
    A [ID] [StateID] .... [Action Statement]
Transition Line:
    W [ID] [Priority] [SrcStateID] [DstStateID]
Condition:
    C [ID] [TranLineID]....[Condition Expression]

```

Fig 3.1 ASF File Format

In the State Object, the information whether it is set as Default state or Trap state is stored in LSB 2 bits in the *[isDefStat/isTrapState]* Field. The LSB bit stands for isDefState and its next bit stands for isTrapState.

In the Transition Line object, the Priority information is stored in 12 bits starting from LSB in the *[Priority]* field.

A line in the ASF file looks like the following:

```
A 25 10 4 TEXT "Actions" | 94596,185504 1 0 0 "D=0;"
```

An ASF file contains each objects graphical information. They are used to show the FSM graphically in the Active HDL State Editor. But they do not have any use in generating SystemC code. So, we ignore those information.

4 Tool Structure

In this section, we describe the developed tool which generates SystemC code out of an FSM description. The block diagram of our tool is shown in Fig 4.1

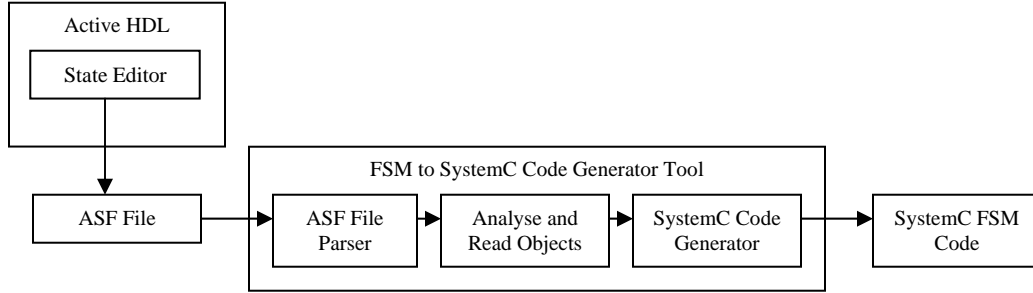


Fig 4.1 SystemC Code Generator Tool

The Parser is used to parse the information of ASF file. Then the ASF file objects and their properties are read in the data structures as shown in Fig 4.2

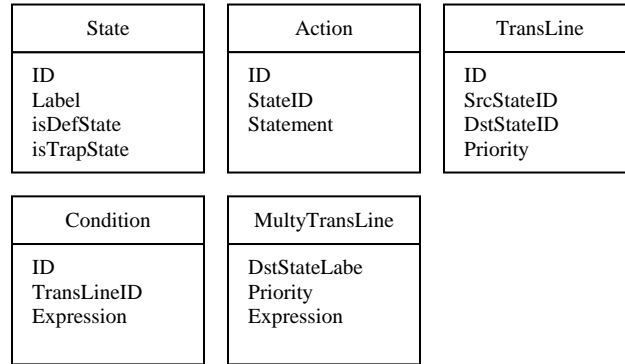


Fig 4.2 Data Structures for FSM Objects and their Properties

After that, SystemC Code Generator generates SystemC code by reading the data structure. The SystemC code generation algorithm is shown in Fig 4.3

```

for each State
{
  Write "Case:" State.Label

  for each Action
    if ( Action.StateID == State.ID )
      Write Action.Statement
  
```

```

for each TransLine
  if ( TransLine.SrcStateID == State.ID )
  {
    new MultyTransLine
    MultyTransLine.Priority := TransLine.Priority
    MultyTransLine.DstStateLabel:=
      ToLabel(TransitionLine.DstStateID)
    for each Condition
      if ( Condition.ID == TransLine.ID )
        MultyTransLine.Expression := Condition.Expression
        MultyTransLine.isConditional := true

    if Condition not found
      MultyTransLine.isConditional := false
  }

Sort MultyTransLine on Priority in Ascending order

for each MultyTransLine
  Write "if/else if" MultyTransLine.Expression
  Write "NextState = " MultyTransLine.DstStateLabel

  if exist DefState and For all ( MultyTransLine.isConditional == true )
    Write "else NextState = " DefaultState.Label
  }

if exist TrapState
  Write "default:"
  Write "NextState = " TrapState.Label

```

Fig: 4.3 SystemC Code Generation Algorithm

The core FSM code is generated in a `switch...case` block. Two state variables `CurrentState` and `NextState` are used to hold the present state and the state of the next clock cycle, respectively. For a *State*, the code generator writes its *Label* after a `case` statement. Then the *Action Statements* associated with the state are written. Thereafter, the code generator gathers all the *TransLine* and *Condition* informations of that state in an object array named *MultyTransLine*. If there are more than one *TransLine* coming out from the state, then the information of *MultyTransLine* is sorted based on the assigned priority in ascending order. Then the conditions for determining `NextState` are written using `if/else if` statements. If any state is set as *Default state* and there exists no unconditional *TransLine* then assigning *Default state* to `NextState` is done using an `else` statement. The *Trap state* is assigned as `NextState` in the `default` section of the `switch...case` block.

Now, let's go through an example to illustrate the idea. The block shown in Fig 4.4 has two operand inputs a, b. Depending on the op input, it performs add, sub, multiply and division. It also has a start S input. We assert S to start the operation. When operation is done, the output is available at C and it indicates the end of operation by asserting the done D signal.

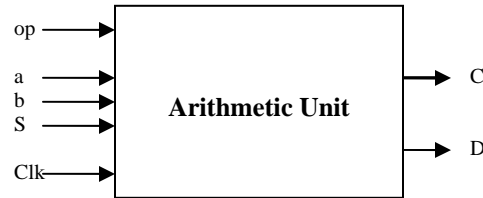


Fig 4.4 Synchronous Arithmetic Unit

We draw its behavior in Active HDL State Editor and then generated the ASF file. The graphical representation of the FSM is shown in Fig 4.5. In Fig 4.5, *S_Init* is set as initial state. Also it is made as the *Default state*. From state *S_OP* four Transition Lines comes out and we assigned priorities on them (shown inside circles). *S_Done* is set as *Trap state*.

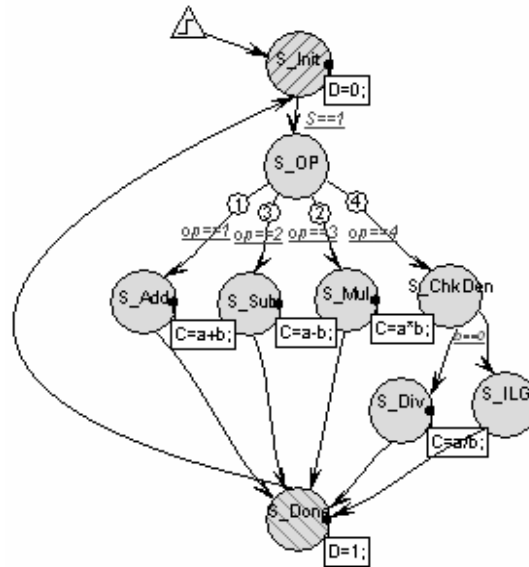


Fig 4.5 Graphical FSM Representation

A portion of the generated ASF File is shown in Fig 4.6

```
L 15 16 0 TEXT "State Labels" | 86000,144896 1 0 0 "S_Sub"
S 14 5 8192 ELLIPSE "States" | 58552,146024 6500 6500
L 13 14 0 TEXT "State Labels" | 58552,146024 1 0 0 "S_Add"
S 12 5 4096 ELLIPSE "States" | 89384,170840 6500 6500
L 11 12 0 TEXT "State Labels" | 89384,170840 1 0 0 "S_OP"
S 10 5 1 ELLIPSE "States" | 88068,191144 6500 6500
L 9 10 0 TEXT "State Labels" | 88068,191144 1 0 0 "S_Init"
```



```

F 5 0 671088652 8 0 "" 0 RECT 0,0,0 0 0 1 255,255,255 0 | 15700,15700 200200,221700
L 6 5 0 TEXT "Labels" | 18700,218700 1 0 0 "Sreg0"
G 1 0 0 TEXT 0,0,0 0 0 0 255,255,255 0 3527 1480 0000 0 "Arial" 0 | 97950,263700 1 0 0
"Entity: des2\nArchitecture: des2_arch"
L 7 8 0 TEXT "Labels" | 26700,227700 1 0 0 "clk"
I 8 0 3 Builtin InPort | 20700,227700 "" ""
A 31 16 4 TEXT "Actions" | 91024,141136 1 0 0 "C=a-b;"
A 30 14 4 TEXT "Actions" | 62824,142264 1 0 0 "C=a+b;"
W 29 5 8196 12 20 BEZIER "Transitions" | 95227,167994 107071,162730 126902,152628
138746,147364
W 28 5 8194 12 18 BEZIER "Transitions" | 93773,166047 98285,161253 106402,154063
110914,149269
W 27 5 8195 12 16 BEZIER "Transitions" | 87971,164498 87078,160503 86584,155380
85691,151385
W 26 5 8193 12 14 BEZIER "Transitions" | 84270,166830 77925,162224 69566,155152
63221,150546
A 25 10 4 TEXT "Actions" | 94596,185504 1 0 0 "D=0;"
S 24 5 28672 ELLIPSE "States" | 158944,117824 6500 6500
L 23 24 0 TEXT "State Labels" | 158756,119140 1 0 0 "S_ILG"
S 22 5 24578 ELLIPSE "States" | 98596,92444 6500 6500
L 21 22 0 TEXT "State Labels" | 98596,92444 1 0 0 "S_Done"
S 20 5 20480 ELLIPSE "States" | 144280,143956 6500 6500
L 19 20 0 TEXT "State Labels" | 144280,143956 1 0 0 "S_ChkDen"
S 18 5 16384 ELLIPSE "States" | 115140,144332 6500 6500
L 17 18 0 TEXT "State Labels" | 115140,144332 1 0 0 "S_Mul"
S 16 5 12288 ELLIPSE "States" | 86000,144896 6500 6500
C 46 45 0 TEXT "Conditions" | 151048,132864 1 0 0 "b==0"
W 45 5 0 20 24 BEZIER "Transitions" | 147668,138410 150347,134227 153646,127953
156325,123770
A 44 41 4 TEXT "Actions" | 136896,112936 1 0 0 "C=a/b;"
W 43 5 0 41 22 BEZIER "Transitions" | 126400,113565 119820,108677 110437,101148
103857,96260
W 42 5 0 20 41 BEZIER "Transitions" | 141673,138002 138994,133819 136502,127999
133823,123816
S 41 5 32768 ELLIPSE "States" | 131308,117824 6500 6500
L 40 41 0 TEXT "State Labels" | 131308,117824 1 0 0 "S_Div"
C 39 34 0 TEXT "Conditions" | 79232,182308 1 0 0 "S==1"
A 38 22 4 TEXT "Actions" | 103432,86616 1 0 0 "D=1;"
W 37 5 0 18 22 BEZIER "Transitions" | 113533,138038 109773,127275 104452,109359
100692,98596
W 36 5 0 16 22 BEZIER "Transitions" | 87495,138571 90221,127949 94464,109411 97190,98789
W 35 5 0 14 22 BEZIER "Transitions" | 61601,140285 70766,128441 84910,108956 94075,97112
W 34 5 0 10 12 BEZIER "Transitions" | 88948,184704 88854,182213 89085,179815 88991,177324
A 32 18 4 TEXT "Actions" | 120352,140384 1 0 0 "C=a*b;"
W 55 5 0 54 10 BEZIER "Transitions" | 71712,198288 75002,196831 78802,195155 82092,193698
I 54 5 0 Builtin Reset | 71712,198288
W 53 5 0 22 10 BEZIER "Transitions" | 96816,98694 31909,122147 29439,157274 87484,184675
C 52 29 0 TEXT "Conditions" | 114200,158996 1 0 0 "op==4"
C 51 28 0 TEXT "Conditions" | 103108,157116 1 0 0 "op==3"
C 50 27 0 TEXT "Conditions" | 86752,158620 1 0 0 "op==2"
C 49 26 0 TEXT "Conditions" | 75660,159936 1 0 0 "op==1"
W 48 5 0 24 22 BEZIER "Transitions" | 153325,114558 139789,109999 118152,99452
104616,94893

```

Fig 4.6 ASF File representation for the Arithmetic Unit

The ASF file generated by Active HDL is then fed to the SystemC FSM code generator tool which parses and analyzes the ASF file and generates SystemC code. A portion of the generated SystemC code is shown in Fig 4.7

```

SC_MODULE ( clsArith )
{
    sc_in <int> a ;
    sc_in <int> b ;
    sc_in <int> op ;
    sc_in <int> S ;
    sc_out <int> C ;
    sc_out <int> D ;
    sc_in <bool> clk ;
    void fsm_run ( void ) ;

    SC_CTOR (clsArith)
    {
        SC_METHOD ( fsm_run ) ;
        sensitive << clk.pos() ;
        InitialState = S_Init;
        CurrentState = InitialState ;
    }
private:
    enum typeState={S_Init,S_OP,S_Add,S_Sub,
    S_Mul, S_Div, S_ILG, S_Done};

    typeState CurrentState,InitialState,
    NextState;
} ;

void clsArith :: fsm_run ( void )
{
    Switch (CurrentState)
    {
        case S_Add:

            //Action
            C=a+b;
            //Next State
            NextState = S_Done ;
            break ;

        case S_OP:
            //Action
            //Next State
            if ( op==1 ) NextState = S_Add ;
            else if (op==3) NextState = S_Mul ;
            else if (op==2) NextState = S_Sub ;
            else if (op==4) NextState = S_ChkDen ;
            else NextState = S_Init ;
            break ;

        case S_Init:
            //Action
            D=0;
            //Next State
            if ( S==1 ) NextState = S_OP ;
            else NextState = S_Init ;
            break ;

        case S_ILG:
            //Action
            //Next State
            NextState = S_Done ;
            break ;

        case S_Done:
            //Action
            D=1;
            //Next State
            NextState = S_Init ;
            break ;

        case S_ChkDen:
            //Action
            //Next State
            if (b==0) NextState = S_ILG ;
            else NextState = S_Div ;
            break ;

        case S_Mul:
            //Action
            C=a*b;
            //Next State
            NextState = S_Done ;
            break ;

        case S_Sub:
            //Action
            C=a-b;
            //Next State
            NextState = S_Done ;
            break ;

        case S_Div:
            //Action
            C=a/b;
            //Next State
            NextState = S_Done ;
            break ;

        //Trap State, the state machine will
        //terminate on any illegal state
        default:
            NextState = S_Done ;
    } //switch

    //update Current State
    CurrentState = NextState ;
}

```

Fig 4.7 Generated SystemC Code

The FSM code is generated in a `switch...case` format [21]. Two state variables `CurrentState` and `NextState` are used to hold the present state and the state of the next clock cycle, respectively. Depending on the `CurrentState`, the state machine reaches its assigned state. Then *Action Statements* of that state are executed. Thereafter `NextState` is determined depending on the *Transition Line Conditions* and their assigned *Priorities*. *Default State* is set as the `NextState` if no *Transition Line Conditions* are satisfied. Then it updates `CurrentState` with the state of `NextState`. At next clock cycle, depending on the `CurrentState` it reaches its assigned state and so on. On any illegal assignment of `NextState`, the state machine reaches the state which is set as *Trap State*.

Default state is made the next state if no Transition Line condition is satisfied as shown in state `S_Init`. If the state has any unconditional Transition Line, then assigning Default state to next state is omitted as shown in case `S_Add`.

At state `S_OP`, we see that the next state conditions appear in the generated SystemC code according to the assigned priority. Consider the following situation where, `a` and `b` are two inputs for a FSM.

```
if (a==1) and (b==1) NextState = s1
else if (a==1) NextState = s2
```

if both `a` and `b` are 1, then `NextState` is dependent on the order of the appearance of the conditions in the code.

This FSM code can be used in both *Method* and *Thread* process of SystemC [21]. For *Method* process, the FSM codes are placed inside a member function and the function is made sensitive to clock signal positive or negative edge. For *Thread* process, the FSM codes are written inside an infinite loop and a wait statement like `wait (clk->posedge_event())` is placed after the `switch...case` block.

5 Conclusion

In this report, we have presented a flow that user can draw an FSM and then our tool can extract SystemC code from it. We have implemented this tool in Microsoft Windows environment. Similar industrial tools that generate VHDL and Verilog code are available, but generating SystemC code from graphical user interface was missing and this work fulfills this gap. We believe our tool can be very useful to convert FSM to SystemC and thus help in analysis and simulation.

Reference

- [1] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo, "A Technique for Drawing Directed Graph", IEEE Transactions on Software Engineering, 19 (3), 214-230, 1993.
- [2] Active HDL State Editor,
<http://support.aldec.com/KnowledgeBase/Article.aspx?aid=000604&show=Avh00089.htm>, 2006
- [3] Active HDL State Editor Training Guide. <http://www.sss-mag.com/pdf/stedtut.pdf>, 2006
- [4] Open SystemC Initiative. www.systemc.org, 2006
- [5] F. van Ham, H. van deWetering, and J. J. vanWijk, "Interactive Visualization of State Transition Systems", IEEE Transactions on Visualization and Computer Graphics, 8(4), 319-329, 2002
- [6] M. Frohlich, and M. Werner, "Demonstration of the interactive Graph Visualization System daVinci", In Graph Drawing of Volume 894, Lecture Notes in Computer Science, Springer Verlag, pp 15-22, 1995.
- [7] VCG Graph Visualization, www.cs.unisb.de/RW/users/sander/html/gsvcg1.html, Universitat des Saarlandes, Germany, 1996.
- [8] Absint Inc., <http://www.absint.com/index.html>, 2003
- [9] Xilinx ISE Tools, http://www.xilinx.com/ise/design_tools/, 2003.
- [10] IEEE Standard 1076-1993, IEEE Standard Description Language Based on the VHDL Hardware Description Language, 1993.
- [11] IEEE Standard 1364-2001, IEEE Standard Description Language Based on the Verilog Hardware Description Language, 2001.
- [12] IEEE Standard 1666-2005, IEEE Standard Description of SystemC Language, 2005.
- [13] Creating_HDL_Graphical_Modules.
http://www.aldec.com/training/active-hdl/pdf/C02_Creating_HDL_Graphical_Modules.pdf, 2006
- [14] Aldec Active HDL. <http://www.aldec.com>, 2006
- [15] E. M. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis". Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, USA, Technical Report CA 94720, 1992.
- [16] C. Pruteanu, "Kiss to Verilog FSM Converter", codrin.freeshell.org, 2000.
- [17] M. Himsolt, "GraphEd: A Graphical Platform for the Implementation of Graph Algorithms", In Graph Drawing of Volume 894, Lecture Notes in Computer Science,

Springer Verlag, pp 182-193, 1995.

- [18] Software Visualization Tools, <http://www.ece.pdx.edu/~alanmi/software/>, Portland State University, USA, 2000.
- [19] VGJ Tool: [www.eng.auburn.edu/departement/cse/research/graph drawing/graph drawing.html](http://www.eng.auburn.edu/departement/cse/research/graph%20drawing/graph%20drawing.html), Auburn University, USA, 1998.
- [20] A. T. Abdel-Hamid, M. Zaki and S. Tahar, A Tool Converting Finite State Machine to VHDL. Canadian Conference on Electrical and Computer Engineering, May 2004, Niagara Falls, Canada
- [21] T. Grotker, S. Liao, G. Martin, and S. Swan. System design with SystemC. Kluwer Academic Publishers, 2002.

Appendix A

Tool User Interface and Source Code

The tool is implemented in Microsoft Visual Basic 6. It has the user interface shown at Fig. A.1

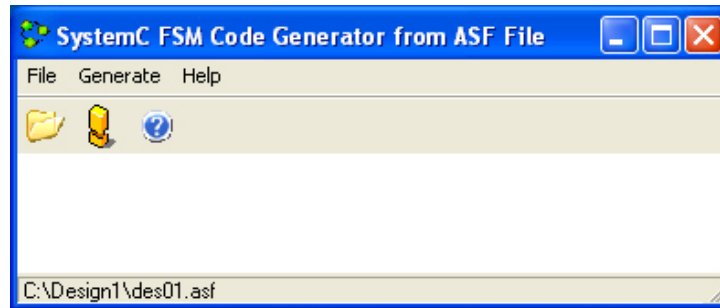


Fig. A.1 Tool Graphical User Interface

Using the tool Graphical User Interface (GUI), the user can browse the ASF file as input to the tool. When Generate SystemC code icon is clicked, the tool prompts the user to select a filename where the SystemC code will be saved. Then the tool generates the SystemC code according to the SystemC FSM code generation algorithm.

Tool Source Code

```
Public Sub WriteSystemC_FSM()
On Error Resume Next

'write typestate variables
Print #1, Space(8) & "//State Variables"
Print #1, Space(8) & "typeStateTrans CurrentState , NextState , InitState, EndState ;"
Print #1, ""

Print #1, Space(8) & "//Set Init State"
Print #1, Space(8) & "InitState = " & State(InitStateIndex).Label & " ;"
Print #1, ""

Print #1, Space(8) & "//Set End State"
Print #1, Space(8) & "EndState = " & State(TrapStateIndex).Label & " ;"
Print #1, ""

Print #1, Space(8) & "//Set Current State"
Print #1, Space(8) & "CurrentState = InitState ;"
Print #1, ""

'write fsm
Print #1, Space(8) & "//FSM"
Print #1, Space(8) & "while ( CurrentState != EndState )"
Print #1, Space(8) & "{"
Print #1, ""

Print #1, Space(16) & "Switch (CurrentState)"
Print #1, Space(16) & "{"

For si = 1 To TotalState
'write case
Print #1, Space(16) & "case " & State(si).Label & ":"
```

```

Print #1, ""

'write actions for this state
Print #1, Space(24) & "//Action"
For ai = 1 To TotalAction

    If Action(ai).StateID = State(si).ID Then

        'if more than 1 action then we have to split
        ActionIArray = Split(Action(ai).Action, "\n")
        Err = False
        For i = 0 To MAX_ACTION_IN_A_STATE - 1
            If Err = False Then
                Print #1, Space(24) & ActionIArray(i)
            Else
                Exit For
            End If
        Next i

        Print #1, ""
    End If

Next ai

'find transition lines for this state
'clear total
TotalMultyTranLineInfo = 0
For ti = 1 To TotalTranLine

    If TranLine(ti).SrcStateID = State(si).ID Then
        'tran line found
        'incr total
        TotalMultyTranLineInfo = TotalMultyTranLineInfo + 1

        'set des state id
        MultyTranLineInfo(TotalMultyTranLineInfo).DstStateID = TranLine(ti).DstStateID

        'set priority
        MultyTranLineInfo(TotalMultyTranLineInfo).Priority = TranLine(ti).Priority

        'clear isconditional flag
        MultyTranLineInfo(TotalMultyTranLineInfo).isConditional = False

        'find condition
        For ci = 1 To TotalCondition

            If Condition(ci).TranLineID = TranLine(ti).ID Then
                'condition
                MultyTranLineInfo(TotalMultyTranLineInfo).Condition =
Condition(ci).Condition

                'set isconditional flag
                MultyTranLineInfo(TotalMultyTranLineInfo).isConditional = True
            End If
        Next ci

    End If

Next ti

'fill dststatelabel property
For i = 1 To TotalMultyTranLineInfo
    For j = 1 To TotalState

        If MultyTranLineInfo(i).DstStateID = State(j).ID Then

            MultyTranLineInfo(i).DstStateLabel = State(j).Label

```

```

        End If

    Next j
Next i

'sort MultyTranLineInfo by priority , ascending order
For i = 1 To TotalMultyTranLineInfo - 1
    For j = i + 1 To TotalMultyTranLineInfo

        If MultyTranLineInfo(i).Priority > MultyTranLineInfo(j).Priority Then

            'swap contents
            MultyTranLineInfo(0) = MultyTranLineInfo(i)
            MultyTranLineInfo(i) = MultyTranLineInfo(j)
            MultyTranLineInfo(j) = MultyTranLineInfo(0)

        End If

    Next j
Next i

'write next state
Print #1, Space(24) & "//Next State"

'set write def state flag
isWriteDefState = True

For i = 1 To TotalMultyTranLineInfo

    'Print #1, MultyTranLineInfo(i).Priority

    '1st condition only if
    If i = 1 Then
        If MultyTranLineInfo(i).isConditional = True Then
            'write if
            Print #1, Space(24) & "if ( " & MultyTranLineInfo(i).Condition & " )
NextState = " & MultyTranLineInfo(i).DstStateLabel & " ;"

        Else 'unconditional, it must be the only tran line
            'write no condition
            Print #1, Space(24) & "NextState = " & MultyTranLineInfo(i).DstStateLabel & "
;"

            'reset isWriteDefState flag
            isWriteDefState = False

        End If

    Else 'i > 1

        If MultyTranLineInfo(i).isConditional = True Then
            'write elseif
            Print #1, Space(24) & "else if ( " & MultyTranLineInfo(i).Condition & " )
NextState = " & MultyTranLineInfo(i).DstStateLabel & " ;"

        Else 'unconditional, it must be last tran line according to priority
            'write else
            Print #1, Space(24) & "else NextState = " &
MultyTranLineInfo(i).DstStateLabel & " ;"

            'reset isWriteDefState flag

```



```

        isWriteDefState = False

    End If

End If

Next i

'write default state, if there is any and whether we should write it
If DefStateIndex > 0 And isWriteDefState = True Then
    Print #1, Space(24) & "else NextState = " & State(DefStateIndex).Label & " ;"
End If

'write break
Print #1, Space(24) & "break ;"

'gap to start code for next state
Print #1, ""
Print #1, ""

Next si

'write trap state
Print #1, Space(16) & "//Trap State, the state machine will terminate on any illegal
state"
Print #1, Space(16) & "default:"
Print #1, Space(24) & "NextState = " & State(TrapStateIndex).Label & " ;"
Print #1, ""
Print #1, ""
Print #1, Space(16) & "} //switch"

'write clock edge event
Print #1, ""
Print #1, ""
Print #1, Space(16) & "//wait for clock positive event"
Print #1, Space(16) & "//so the states changes at each clk posevent"
Print #1, Space(16) & "wait (clk->posedge_event() ) ;"

'write update state
Print #1, ""
Print #1, ""
Print #1, Space(16) & "//update Current State"
Print #1, Space(16) & "CurrentState = NextState ;"

Print #1, ""
Print #1, ""
Print #1, Space(8) & "} //while"
End Sub

```