

A Tool Converting Finite State Machine to VHDL

Amr T. Abdel-Hamid, Mohamed Zaki and Sofiène Tahar
Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montreal, Quebec, H3H 1M8, Canada
Email: {at_abdel, mzaki, tahar}@ece.concordia.ca

Abstract

Finite state machines (FSM) are a basic component in hardware design, 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, 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 paper, we present a tool which starting from a graphical FSM representation, produces a behavioral HDL code which can be directly analyzed and synthesized.

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 this project, starting from a visualization tool Graphviz [7] allowing users to interactively explore large state spaces, we extract HDL code which can be used in simulation and synthesis. The state machine design is converted to a state table

and then, into VHDL description. The Graphviz is used as a graph editor for drawing the state transition graph (STG) of the design required. Graphviz outputs a *dot* language [7] file that gives a textual representation for the FSM machine. The proposed tool converts this file first to Kiss2 format [9], which is a standard FSM format that is used by many tools. This Kiss2 file is used by the other part of the tool to directly generate sequential VHDL code.

In the next sections, we will discuss relevant related work concerning visualization languages, FSM based languages as well as the most popular HDLs on the market today. Then, we will describe the proposed tool in details, and finally we are going to give an example of one file converted from FSM to VHDL by our tool.

2 Related Work

2.1 Interface Languages and Visualization Systems

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, dataflows, and so on. Usually graph visualization tools support the following options: directed, undirected, and mixed graphs, hypergraphs, hierarchical graphs, graphical representations [11]. Different format have been proposed as input to the visualization tool. 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. Among the most important is the *dot* format described in this paper. Other format include GraphML (GML) [4] which is an upcoming graph specification standard. Among the tools using GML is GraphLet [4], implemented in C++. It provides a scripting language to support user interfacing and animation tailoring of the graph editor. VGJ [13] is also a graph layout tool based on GML. It is written in Java and includes techniques for hierarchical directed graphs. The tool supports 3D and file input/output in GML.

Other visualization tools include the daVinci graph visualization [3] program and VCG tool [12] which automatically computes the most optimal way to view the finite-state automaton by minimizing the number of crossing edges. AiSee [1], which is a part of the Absint static analyzer tool suite [1], 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) [3]. 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 [14] include an editor, named StateCAD, which allows users to graphically input state diagrams and have it translated into a Verilog behavioral HDL model. Non-commercial tools includes Visual Software (SYNTHA) [10] where the input is an FSM specification using an autogram, a Kiss [2] table or (in the future) VHDL and the output is an encoded, minimized, and mapped netlist of gates, which implements the given FSM. A variety of options allow the user to select encoding strategies and flip-flop types.

2.2 Modelling Languages

Among existing FSM modeling languages, we have chosen the Kiss format [2]. Its FSM description assumes symbolic names for the state (pre-encoding), while inputs and outputs are specified using the three symbols 1, 0, and - (don't care). Kiss is a tabular format, where each row has four entries: input field, present state field, next state field and output field. There are as many rows as transitions in the state graph of the FSM. BLIF [9] (Berkeley Logic Interchange Format) was developed to describe a logic-level circuit in textual form. A circuit is a combinational or sequential network of logic functions. It can be viewed as a directed graph of combinational logic nodes and sequential logic elements. Each node has a single-output logic function associated with it. Each feedback loop must contain at least one latch (flip-flop). Each net (or signal) has only a single driver, and either the signal or the gate which drives the signal can be named without ambiguity.

2.3 HDL Languages

Modern hardware designers typically uses hardware description languages (HDLs) to express 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, con-

currency and data structure suitable for modelling different aspects of hardware. The most popular hardware description languages are VHDL [5], and Verilog [6]. We have chosen VHDL in this project.

VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) [5] is an IEEE Standard since 1987 while Verilog was only standardized in 1995. Both languages are programming language that has been designed and optimized for describing the behavior of digital systems, they support the development, verification, synthesis, and testing of hardware designs. Like C and C++, VHDL and Verilog include features useful for structured design techniques, and offer a rich set of control and data representation features.

3 Tool Structure

In this section, we describe the developed tool which generates VHDL behavioral code out of an FSM description. The tool was implemented using C++ under Unix environment. It is composed of two different modules interacting together. The advantage of this modularity is the simplicity of updating its architecture by modifying the input format, the VHDL subset or enhancing its performance. The tool is basically composed of two parts:

1. A Dot-to-Kiss2 module, that converts the dot file to a kiss2 formatted FSM.
2. A Kiss2-to-VHDL module, that uses the generated kiss2 file to generate the VHDL description of the design.

3.1 DOT-to-Kiss2

Figure 1 shows the architecture of the DOT-to-Kiss2 module which transform a *dot* representation file of an FSM design into Kiss2 format. The first two modules parse the dot file, analyze it to check for possible syntax violation or errors and then create a parse tree. The parse tree is then input to a Kiss2 Generator which create at the back end a kiss format. A transition in dot format has the following form:

```
current_state -> Next_state [label =


```

The translation from the dot format to the Kiss2 format is straight forward as transitions and states are directly identified. The *or* in the transition description means that several input/output combinations could lead to the same transition.

As an illustrative example, we consider one of the LGSynth93 benchmark FSMs [8] in Figure 2, called *lion*.

The lion FSM has four different states, two inputs and one output. Each of these states has four transitions as shown in Figure 2(a). Figure 2(b) shows the output generated by the Graphviz tool in dot format. Figure 3 shows the output of the first stage of the tool after converting the dot file to Kiss2 standard representation.

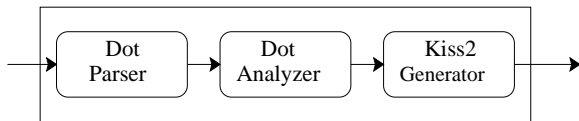
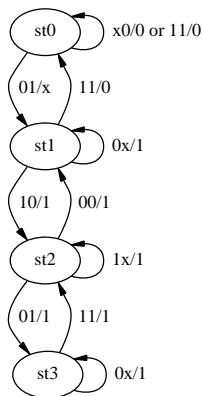


Figure 1. Dot-2-Kiss2 Block Architecture



(a)

```
digraph EX1 {
  0 [label = "st0" ]
  1 [label = "st1" ]
  2 [label = "st2" ]
  3 [label = "st3" ]
  0 -> 0 [label = "x0/0 or 11/0" ]
  0->1[label = "01/x" ]
  1 -> 1 [label = "0x/1" ]
  1 -> 0 [label = "11/0" ]
  1 -> 2 [label = "10/1" ]
  2 -> 2 [label = "1x/1" ]
  2 -> 1 [label = "00/1" ]
  2 -> 3 [label = "01/1" ]
  3 -> 3 [label = "0x/1" ]
  3 -> 2 [label = "11/1" ]
}
```

(b)

Figure 2. Lion FSM and its dot Representation

```
.i 2
.o 1
.p 11
.s 4
-0 st0 st0 0
11 st0 st0 0
01 st0 st1 -
0- st1 st1 1
.
.
```

Figure 3. Generated Kiss2 Code for the Lion FSM

3.2 Kiss2-to-VHDL

Figure 4 shows the architecture of the Kiss2-to-VHDL converter. The converter is composed of three blocks: Kiss2 parser, Kiss2 analyzer, and VHDL generator.

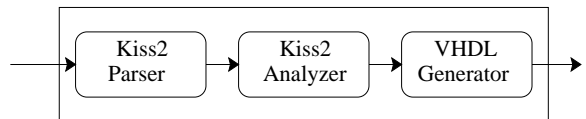


Figure 4. Kiss2-to-VHDL Block Architecture

The Kiss2 parser takes the Kiss2 file as its input. Figure 3 shows a sample Kiss2 file, which starts by four lines specifying the different attributes. These attributes are: the number of inputs (i), the number of outputs (o), the number of transitions (p), and the number of states (s). These attributes are followed by the system description stating all the transitions of the system starting by the input then the current state, the next state and the associated output.

The Kiss2 analyzer is the main component of this part of the tool part. It accept the output of the analyzer and use it to build the FSM tree that will be used to generate the VHDL code afterwards. This tree includes the number of transitions associated with each state to ease the VHDL code generating in the next step. This module generates also the names of the states, and if the states are defined using binary bits, it changes this to integer numbers initiated with the two letters 'st' to make it possible for the VHDL generator to generate the VHDL code directly.

The VHDL generator takes the number of inputs, the number of outputs, the number of transitions, as well as the number and names of the states. The module then defines the different aspects needed in a VHDL file, it starts by generating the name of the module as well as the different associated numbers in VHDL format. Finally, it generates different transitions of the design and associates it will dif-

ferent states by using the tree defined in the previous module.

For the lion benchmark FSM discussed in the previous section, Figure 5 shows the final representation of the FSM design in VHDL sequential code.

```

ENTITY lion IS PORT(
INPUT1: IN STD_LOGIC_VECTOR(1 DOWNT0 0);
OUTPUT1: OUT STD_LOGIC ) ;
END loin ;

ARCHITECTURE Behavior OF lion
IS TYPE State_type
state(st0,st1,st2,st3);
BEGIN
    PROCESS (INPUT) BEGIN
case state is
    when st0 => if (INPUT1 = 'X1') then
        { OUTPUT1 = '0'; state <= st0 }
        elseif (INPUT1 = '11') then
            {OUTPUT1 = '0'; state <= st0}
            .
            .
        when st1 =>
            .
            .
    END PROCESS ;
END Behavior ;

```

Figure 5. Generated VHDL Code for the Lion FSM

4 Conclusions

We presented in this paper a tool extracting behavioral VHDL code from a graphical representation of FSMs. We have implemented this tool in C++. Similar industrial tools are available, but to our best knowledge not in the open source library. Pruteanu [2] has implement a similar tool that converts Kiss2 files directly to Verilog HDL [6], but it is missing the graphical user interface proposed in our tool. We believe our tool provides a very friendly graphical user interface that can be used very effectively in simulation and synthesis of sequential circuits.

References

- [1] Absint Inc., <http://www.absint.com/index.html>, 2003.
- [2] C. Pruteanu, "Kiss to Verilog FSM Converter", codrin.freeshell.org, 2000.
- [3] M. Frohlich, and M. Werner, "Demonstration of the interactive Graph Visualization System daVinci", In Graph Drawing, Volume 894 of Lecture Notes in Computer Science, Springer Verlag, 15-22, 1995.
- [4] M. Himsolt, "GraphEd: A Graphical Platform for the Implementation of Graph Algorithms", In Graph Drawing, Volume 894 of Lecture Notes in Computer Science, Springer Verlag, 182-193, 1995.
- [5] IEEE Standard 1076-1993, IEEE Standard Description Language Based on the VHDL Hardware Description Language, 1993.
- [6] IEEE Standard 1364-2001, IEEE Standard Description Language Based on the Verilog Hardware Description Language, 2001.
- [7] 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.
- [8] K. McElvain, "LGSynth93 Benchmark Set Version 4.0", http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth93/, 1993.
- [9] 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.
- [10] Software Visualization Tools, <http://www.ece.pdx.edu/~alanmi/software/>, Portland State University, USA, 2000.
- [11] F. van Ham, H. van de Wetering, and J. J. van Wijk, "Interactive Visualization of State Transition Systems", IEEE Transactions on Visualization and Computer Graphics, 8(4),319-329, 2002
- [12] VCG Graph Visualization, www.cs.uni-sb.de/RW/users/sander/html/gsvcg1.html, Universitat des Saarlandes, Germany, 1996.
- [13] VGJ Tool: www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html, Auburn University, USA, 1998.
- [14] Xilinx ISE Tools, http://www.xilinx.com/ise/design_tools/, 2003.