Web Based Tool for State Machines Design

Vassiliy Platonovitch Tchoumatchenko

Abstract – **The paper describes a web based tool for designing finite state machines (FSM). The user specifies a state machine by drawing a diagram, from which the tool generates a synthesizable VHDL model. The intended application of the software is to help student in learning HDL based design of electronic circuits.**

Keywords – **Finite State Machine (FSM), state diagram, VHDL**

I. Introduction

In the context of electronic design automation, a finite-state machine (FSM), or simply a state machine, is a mathematical model used to design sequential logic circuits. It is defined by a list of states, the triggering condition for each transition and the output values. The state machines are convenient formalism for specifying the behavior of control circuits. As such they are often used in the digital circuits design.

FSMs can be represented graphically, which would help the designer to visualize and design in a more efficient way. Most modern digital IC design flows are based on hardware description languages (HDL). Therefore, the designer requires a straightforward way to convert the visualized design to HDL code in order to simulate and implement it. The procedures for such conversions are well defined and can be implemented as a design automation tool. Such tool can be especially helpful for students, who are learning digital design and HDLs.

The task of converting state diagrams to HDL code is addressed be several existing tools, both open-source and commercial – [1], [2], [3] and [4]. What these tools have in common is that all require local installation. Since our goal is to move as much as possible of the student's design works in a web-based collaborative environment, this is a serious drawback.

II. Tool Structure

The tool consists of state diagram editor; JSON store and HDL model generator services (fig. 1). The user interface is implemented in JavaScript and HTML5, which makes it accessible on both mobile devices and desktop computers [5], [6].

The serialized state diagrams are persisted in a JSON store – which can be either a private database installation (e.g. CouchDB) or a cloud based service like GitHub [7].

Once the state diagram is finished, it is sent to the selected HDL generator. At the time of writing, the VHDL generator is already available and a System Verilog generator is under development. The state diagram is analyzed and suitable messages are produced if inconsistencies are found. After the validation, a HDL model is generated and transferred back to the UI. The HDL generators are implemented as web services.

V. Tchoumatchenko is with the Department of Electronics and Electronics Technologies, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski Blvd., 1000 Sofia, Bulgaria, e-mail: vpt@tu-sofia.bg

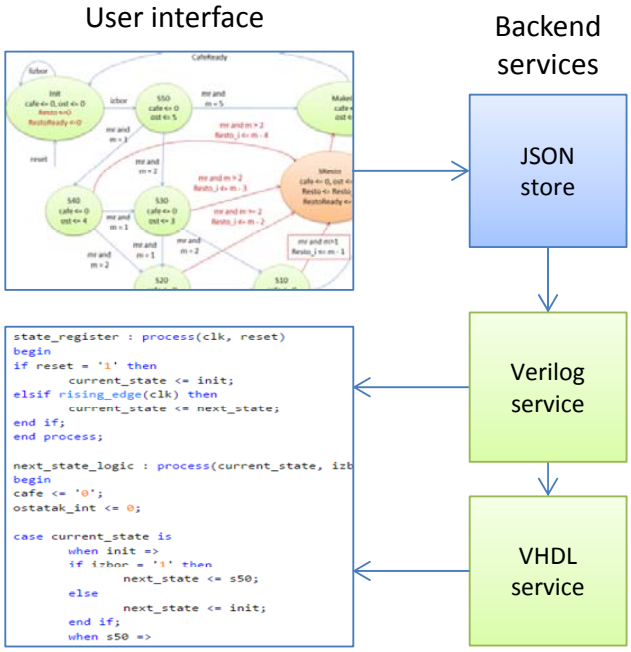

Fig. 1. Components of the FSM tool

III. Tool Functionality

A. State Diagram Editing

The primary design entry method of the FSM tool is a state diagram editor (fig.2). Both Moore and Mealy state machines (fig. 3) are supported. If the output assignments are defined only in the states, than a Moore state machine will be generated. Alternatively, if there is output assignments associated with the transitions, the tool produces model of a Mealy FSM.

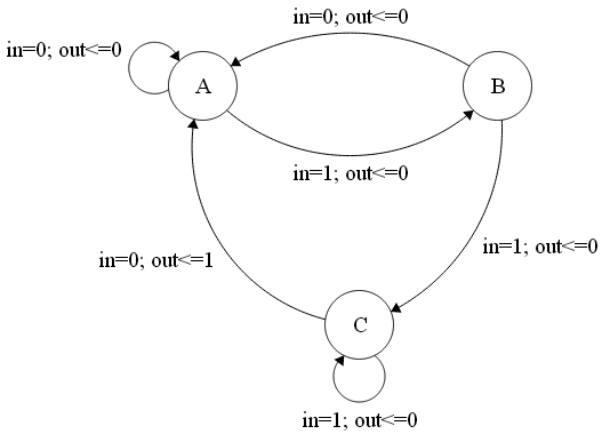

Fig. 2. FSM state diagram

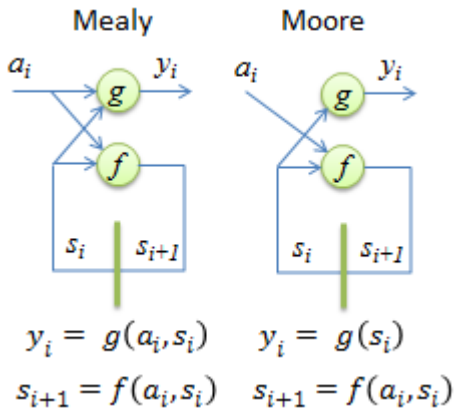

Fig. 3. State machine types: (a) Mealy, (b) Moore

B. HDL Code Generation

By default, the VHDL code generator implements a three-process approach to structuring the FSM code (fig. 4). One sequential process models the state register (fig. 5a) and two combinational processes describe the next state logic (fig. 6) and the output logic (fig. 7). For smaller state machines, the user has an option to choose a two-process architecture, where the two combinational processes are merged into one.

The VHDL generator uses symbolic state enumerations (fig. 5b), so that the actual state encoding can be decided at the synthesis stage. This allows the designer to explore the effect of the state encoding algorithm (e.g. one-hot vs Gray vs binary) on the produced circuit. The next state logic process contains code for recovering from parasitic states (fig. 6a).

```
architecture v1 of sequence_recognizer is
    type fsm_states is (a, b, c);
    signal current_state : fsm_states;
    signal next_state : fsm_states;
begin

state_register: process(clock, reset)
begin
    if(reset = '1')then
      current_state <= a;
    elsif(rising_edge(clock))then
      current_state <= next_state;
    end if;
end process;
```

a

b

Fig. 5. Symbolic state type (a) and state register (b).

```
nsl: process (current_state, input)
begin
  case current_state is
  when a =>
    if(input = '1')then
      next_state <= b;
    else
      next_state <= a;
    end if;
  when b =>
    if(input = '1')then
      next_state <= c;
    else
      next_state <= a;
    end if;
  when c =>
    if(input = '0')then
      next_state <= a;
    else
      next_state <= c;
    end if;
  when others =>
    next_state <= a;
  end case;
end process;
```

a

Fig. 6. Next state logic process.

```
ol: process (current_state, input)
begin
  case current_state is
  when a =>
    output <= '0';
  when b =>
    output <= '0';
  when c =>
    if(input = '0')then
      output <= '1';
    else
      output <= '0';
    end if;
  when others =>
    output <= '0';
  end case;
end process;
```

Fig. 7. Output logic process.

IV. EDUCATIONAL APPLICATIONS

The described tool is include in the HDL based IC design workflow used by some of the 4-th and 5-th year students at the Department of Electronics, TU-Sofia.

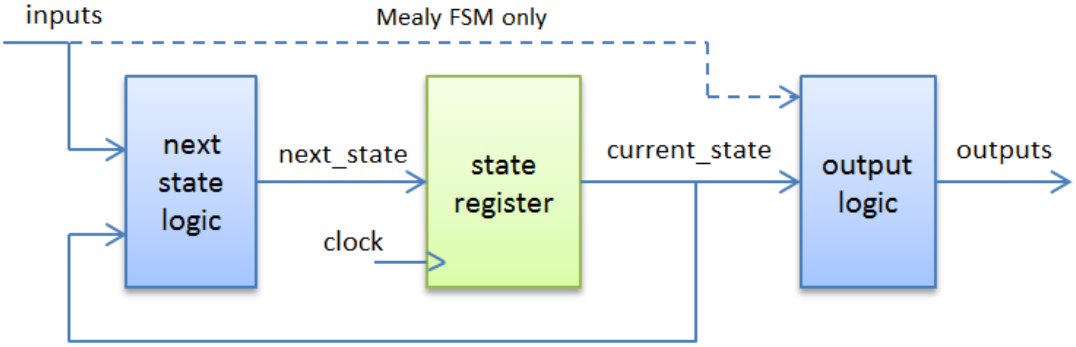

Fig. 4. Three-process FSM architecture.

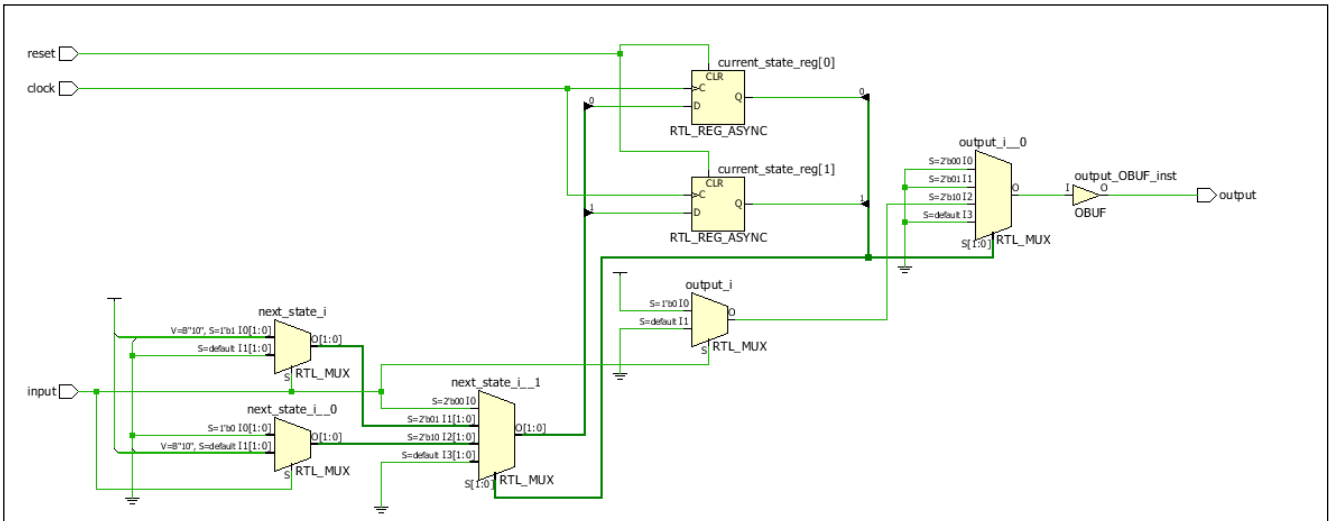

Fig.8. Synthesized circuit (Xilinx Vivado)

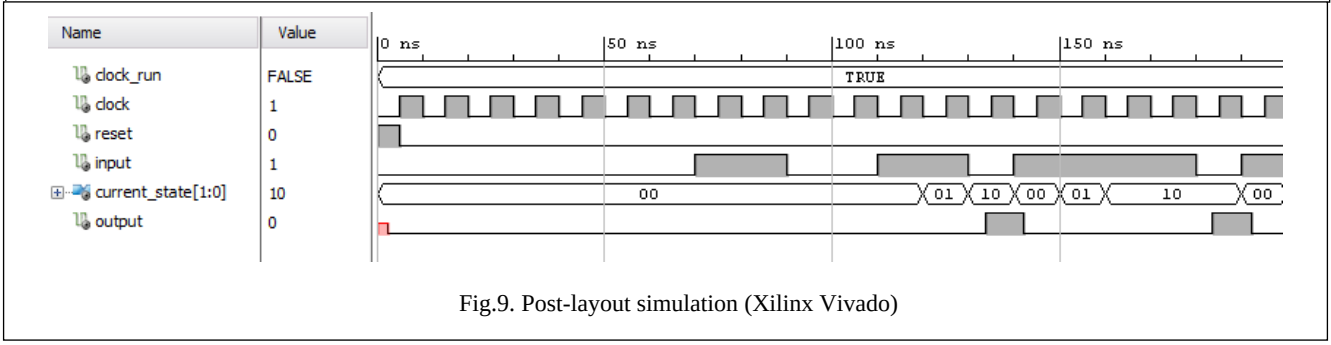

Fig.9. Post-layout simulation (Xilinx Vivado)

The students are required to work on multiple homework assignments and a final project. They can accomplish most of the front-end design (design entry and simulation) and all of the report preparation by using web based tools – EDA Playground [9], Google Docs, and the FSM tool described in this paper.

For the synthesis, physical design and post-layout simulation (fig. 8, 9), the students can use Xilinx Vivado [10]. It is available in the lab or the students can install a free Vivado version on their own computers.

V. Conclusion

The paper considers features, architecture and software implementation of the developed design automation tool for converting state diagrams to VHDL code. The web based nature of the FSM design tool facilitates its integration with similar design automation tools for code editing and HDL simulation. Together they can be used to support on-demand, self-paced learning of digital IC design.

Acknowledgements

The work in this paper is a part of the EU project "Promoting Knowledge Work Practices in Education – KNORK" at the Department of Electronics - Technical University of Sofia, which is supported by the Lifelong Learning Program of the European Community.

References

[1] Amr T. Abdel-hamid and Mohamed Zaki and Sofiène Tahar. *A Tool Converting Finite State Machine to VHDL*, Proc. of IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'04), Niagara Falls, pp. 1907--1910, 2004.

[2] S. Duffner, 'Qfsm - The Finite State Machine Designer', 2015. [Online]. Available: http://qfsm.sourceforge.net/. [Accessed: 04- Jul- 2015].

[3] P. Zimmer, 'Fizzim – an open-source fsm design environment', 2015. [Online]. Available: http://www.fizzim.com/. [Accessed: 04- Jul- 2015].

[4] Xilinx.com, 'StateCAD Help', 2007. [Online]. Available: http://www.xilinx.com/itp/xilinx10/help/iseguide/mergedProjects/state/whnjs.htm. [Accessed: 04- Jul- 2015].

[5] Draw2d.org, 'Draw2D touch', 2015. [Online]. Available: http://www.draw2d.org. [Accessed: 05- Jul- 2015].

[6] Jointjs.com, 'JointJS - the HTML 5 JavaScript diagramming library.' 2015. [Online]. Available: http://www.jointjs.com/. [Accessed: 04- Jul- 2015].

[7] Draw2d.org, 'Github as JSON backend | Draw2D touch', 2015. [Online]. Available: http://www.draw2d.org/draw2d/index_files/65f70d7ec25cfaae0ab862523263d8c4-137.php. [Accessed: 05- Jul- 2015].

[8] Clifford E. Cummings, 'Coding And Scripting Techniques For FSM Designs With Synthesis-Optimized, Glitch-Free Outputs', SNUG-2000, Boston, MA, 2000.

[9] http://www.edaplayground.com/

[10] http://www.xilinx.com/products/design-tools/vivado.html