EE Times Connecting the Global Electronics Community

Blue Pearl Software

1/29/2014

A Kaleidoscopic View of Finite State Machine Design

Srinivasan Venkataramanan, CTO, CVC Pvt Ltd.

Finite state machine (FSM) design at the register transfer level (RTL) of abstraction has always been a fun, challenging, and integral part of digital design. As much fun as FSMs are to conceptualize and design, they can be equally tiring and troublesome to code and get things right in an HDL such as Verilog or VHDL, especially when the machine involves many states and transitions.

FSMs can provide challenges for both RTL designers and verification engineers. Consider an RTL designer who takes over someone else's code. The designer must understand the design, the code, and possible future extensions/maintenance. In such cases, it is apparent that a plain, HDL view can be boring to say the least, and grasping the intent behind the code can be extremely frustrating. For verification engineers, FSMs pose a formidable challenge when it comes to coverage closure to ensure the machine has been thoroughly verified. When was the last time *you* achieved 100% FSM coverage in any real design?

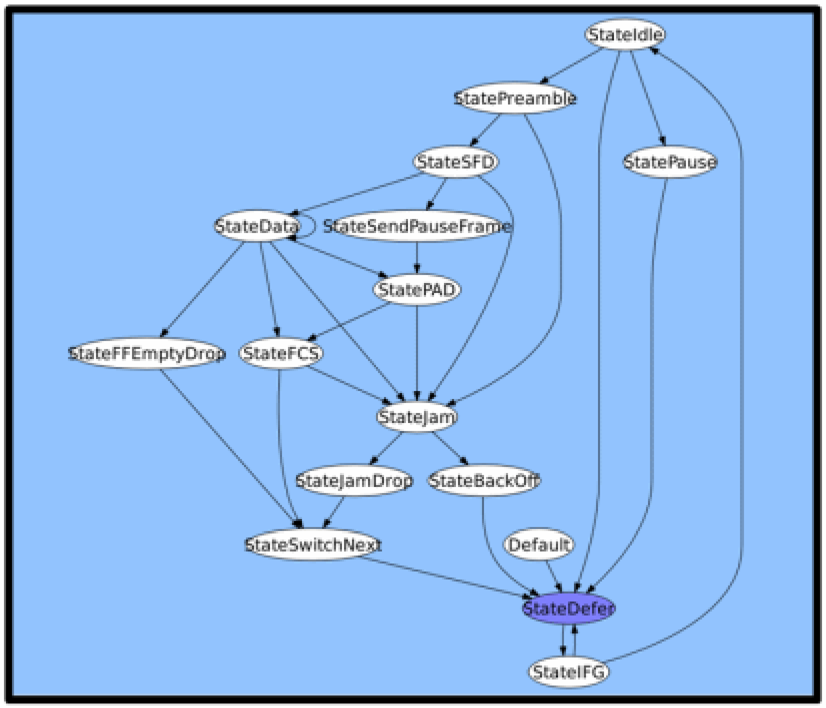
A kaleidoscope is an optical instrument that produces a succession of symmetrical views/colors when rotated. In this article, we will explore an EDA tool that we can visualize as a kaleidoscope for FSM designers, because it provides different views of an FSM design.

Make no mistake; FSMs are a crucial part of many designs, and their implementation can pretty much determine the success or failure of your system and/or its performance. FSMs are slightly different beasts when it comes to designing, coding, verifying, and reviewing. Academic views of FSMs have always been graphical with bubble diagrams as the baseline, but many RTL designers across the globe (perhaps most) do not code using this approach. There are multiple reasons for this, but this has led to serious design errors, flaws, and bugs.

We at TeamCVC have been exploring various approaches to this evergreen phenomenon (a carefully chosen word, since many RTL design experts get FSMs right only after a few iterations). SystemVerilog (SV, IEEE 1800-2012) has emerged as the language of choice for designers and verification engineers, especially those who previously used Verilog. SV supports enumerated types for FSM state definition, custom encoding in the language (unlike a tool-specific pragma/script), struct to capture complex data structures, enhanced procedural blocks (always_ff, always_comb, etc.) to avoid common coding errors, and many other constructs that aid RTL designers. This slideshow offers a detailed look at SV for FPGA RTL design.

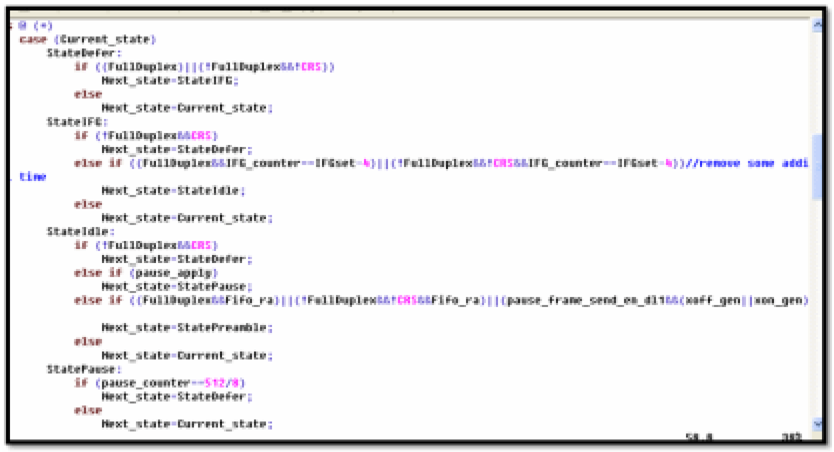
Even with all these advanced constructs, engineers require a lot of visualization capabilities in order to be productive. Recently, TeamCVC looked at Blue Pearl's FSM analysis capabilities. In the following discussions, we consider some of the kaleidoscopic views that Blue Pearl's Analyze RTL product can offer. With every view below, we have annotated our comments on how users can benefit. We refer to this as kaleidoscopic because of the wide variety of views of an FSM that this tool can present.

To start, let's look at a bubble diagram view of a complex, Ethernet MAC controller FSM.



Bubble diagram generated by Bluepearl's Analyze RTL tool. (Click here for a larger image.)

To make sure you appreciate the value of this automatically generated view, consider the raw HDL source code shown below. The value of the bubble diagram is easy to see; it lets you quickly absorb and review the overall machine.



```
@ (*)
case (Current_state)
    StateDefer:
        if ((FullDuplex)||(!FullDuplex&&!CRS))
            Next_state=StateIFG;
        else
            Next_state=Current_state;
    StateIFG:
        if (!FullDuplex&&CRS)
            Next_state=StateDefer;
        else if ((FullDuplex&&IFG_counter==IFGset-4)||(!FullDuplex&&!CRS&&IFG_counter==IFGset-4))//remove some addi
time
            Next_state=StateIdle;
        else
            Next_state=Current_state;
    StateIdle:
        if (!FullDuplex&&CRS)
            Next_state=StateDefer;
        else if (pause_applg)
            Next_state=StatePause;
        else if ((FullDuplex&&Fifo_ra)||(!FullDuplex&&!CRS&&Fifo_ra)||(pause_frame_send_en_dl1&&(xoff_gen||xon_gen)

            Next_state=StatePreamble;
        else
            Next_state=Current_state;
    StatePause:
        if (pause_counter==512/8)
            Next_state=StateDefer;
        else
            Next_state=Current_state;
```

HDL code snippet for the FSM. (Click here for a larger image.)

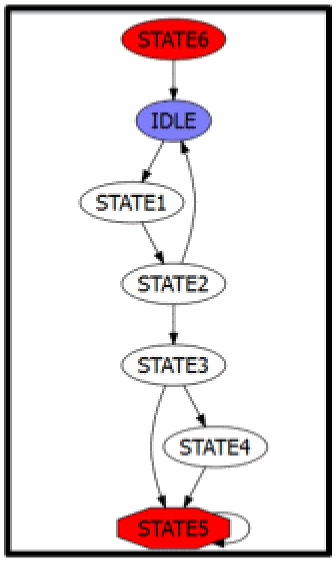
As a good linter, Blue Pearl also performs a CASE statement analysis, in which it will report items such as:

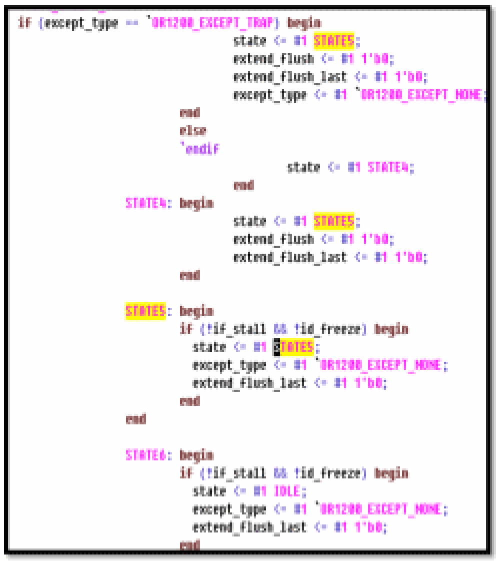
- Duplicate CASE items
- Overlapping CASE items
- Missing CASE items
- Missing assignments in a CASE item
- Missing default clause

This is a good set of automated checks, and such a view/report can save many hours of detailed review. With most of the issues detected and reported, the RTL owner can fix them quickly or provide a reason they exist in the code.

Reachability analysis

One of the most useful views we found in Blue Pearl's tool is its reachability analysis. It will automatically detect FSMs and analyze them structurally. In addition to reporting on all state transitions, the tool will inform you if there are any unreachable or dead states in your state machine. A picture is worth a thousand words. Consider the following bubble diagram screen shot and the associated RTL code. Which view would you prefer for analyzing dead states in your FSM?





```
if (except_type == `OR1200_EXCEPT_TRAP) begin
                                state <= #1 STATE5;
                                extend_flush <= #1 1'b0;
                                extend_flush_last <= #1 1'b0;
                                except_type <= #1 `OR1200_EXCEPT_NONE;
                        end
                        else
                        `endif
                                        state <= #1 STATE4;
                                end
                STATE4: begin
                                state <= #1 STATE5;
                                extend_flush <= #1 1'b0;
                                extend_flush_last <= #1 1'b0;
                        end

                STATE5: begin
                        if (!if_stall && !id_freeze) begin
                          state <= #1 STATE5;
                          except_type <= #1 `OR1200_EXCEPT_NONE;
                          extend_flush_last <= #1 1'b0;
                        end
                end

                STATE6: begin
                        if (!if_stall && !id_freeze) begin
                          state <= #1 IDLE;
                          except_type <= #1 `OR1200_EXCEPT_NONE;
                          extend_flush_last <= #1 1'b0;
                        end
```

Graphical representation of an FSM extracted from the HDL. (Click here for a larger image.)

In addition, the run time for this type of tool is far less than a traditional simulation, and it can be used while the RTL is being developed, without requiring a testbench. It's obvious that linting tools have come a long way. We at TeamCVC are exploring this topic more, and we will address some more features of this technology in a followup article.

Srinivasan Venkataramanan is the chief technology officer at CVC Pvt Ltd, a Verilog and System Verilog training and consulting firm in Bangalore, India. His areas of interest are emerging verification solutions and methodologies such as SystemVerilog, UVM, VMM, OVM, and SoC verification using graph-based scenario models; IP verification using formal methods, and low-power design verification with UPF. He provides support to leading-edge semiconductor design companies on their verification methodologies and challenges. He holds a master's degree in VLSI design from the Indian Institute of Technology in Delhi and a bachelor's degree in electrical engineering from TCE in Madurai. He can be contacted at srini@cvcblr.com.