Experiencing Checkers for a Cache Controller Design

Ben Cohen
VhdICohen Publishing
ben@SystemVerilog.us

Srinivasan Venkataramanan and Ajeetha Kumari
CVC Pvt.Ltd., Bangalore, India
srini@cvcblr.com
akumari@cvcblr.com

Lisa Piper
IC Verification Consultant
lisa@SystemVerilog.us
Purpose of Study

• Are checkers useful?
  • Easy to use?
    – Create, instantiate, Intuitive results, Easy debug

• What was our experience?
  – Process of using checkers
  – Application issues
  – Tool issues
  – Conclusions
Agenda

- Classification of assertions
- Classification of checker instantiations
- The cache controller model
- Example of checker declaration and instantiation
- Design experience
  - Without checkers
  - With checkers
- Impressions and conclusions
Classification of Checkers

Assertions and Instances

Checker assertion type has significance when instantiated

Assertion
- Static
- Procedural

Checker instance
- Static
- Procedural
Emulation of Static Checker Instance (chk1) in a Module

Static instantiation is identical to module instantiation

When a checker is instantiated as a static checker instance, all of its code behaves as if it were instantiated directly in the module after the proper argument associations are made.
Emulation of Procedural Checker Instance in a Module

- Procedural concurrent assertions of checker instanced procedurally behave in-block.
- Static assertions of that checker behave inline to where instanced
The Model – Cache Controller

Cache Controller

- Control Logic
- Cache Memory
- Checker

Main Memory Interface

- mem_rd
- mem_wr
- mem_addr
- mem_rd_data
- mem_wr_data

User interface

- data_vld
- busy
- rd
- wr
- addr
- rd_data
- wr_data
- clk
- rst_n

miss = !cache_exists

cache_exists

cache_data

Many requirements!
Cache Controller Timing
read and write-through cache

Read with data in Cache

Main Memory Read with Cache Miss

User Interface Write
Memory rd/wr access parameterized

Ben Cohen, VhdlCohen Publishing
Cache Controller Architecture

Checker | Function |
---|---|
chk_immd. | Inline assertions, instantiated procedurally |
chk_rd_cntrl. | Read miss/hit, instantiated statically |
chk_rd_wr_cntrl. | RD/WR miscellaneous, instantiated statically |
chk_wr_cntrl. | Write through and cache, instantiated statically |
chk_reset. | Reset, instantiated statically |
chk_fifo | LRU Fifo assertions, instantiated statically |
Cache Memory Architecture

Checker
chk_invalidate.
chk_immd.

Function
Cache invalidates, \textit{instantiated procedurally}
Inline assertions, \textit{instantiated procedurally}

Lots to check and cover!
Example of a Checker Declaration

import cache_ctl_pkg::*;

checker chk_invalidate(
  logic [0: 2**MAIN_MEM_ADDRESS_WIDTH-1] cache_valid,
  logic [MAIN_MEM_ADDRESS_WIDTH-1 : 0]
      [0 : (CACHE_SIZE-1)] cache_mem_addr,
  logic [CACHE_SIZE-1: 0] cache_entry,
  logic [MAIN_MEM_ADDRESS_WIDTH-1:0] addr2invalidate,
  logic found_existing_entry,
  logic clk, rst_n);

timeunit 1ns; timeprecision 100ps;
if (CACHE_SIZE > 1024) $error("cache size is too large");
default clocking default_clk @ (posedge clk); endclocking
default disable iff !rst_n;
Example of a Checker Declaration

function logic check_cache_entry4MT();
  automatic logic success = 0; // If == 1 then it was not invalidated
  automatic int i;
  for (i = 0; i <= CACHE_SIZE - 1; i++) begin : for1
    if (cache_entry[i] == 1 && cache_mem_addr[i] == addr2invalidate) begin : if_1
      success = 1; // found cache line
      break;
    end : if_1
  end : for1
  if (success) check_cache_entry4MT = 0; // was not invalidated
  else check_cache_entry4MT = 1; // was invalidated
endfunction : check_cache_entry4MT

// static concurrent assertion
ap_invalidate : assert property( // Static assertion
  ##1 cache_valid[addr2invalidate] == 0 && check_cache_entry4MT());

// static concurrent assertion
ap_nothing2invalidate: assert property (found_existing_entry)
  else $error("nothing to invalidate");
endchecker : chk_invalidate
Example of a Checker Instantiation (*cache_mem.sv*)

```vhdl
always @(posedge clk) begin : cache_mem_handling
  automatic logic success, found_existing_entry, found_empty_line;
  automatic int i, j, found_index;
  success = 0; found_existing_entry = 0;
  found_index = 0; found_empty_line = 0;
  if (invalidate) begin : if1
    cache_valid[addr2invalidate] <= 1'b0;
    for (int j=0; j<= CACHE_SIZE-1; j=j+1) begin : for1
      if (cache_entry[j] == 1 && cache_mem_addr[j] == addr2invalidate) begin : if2
        found_existing_entry = 1'b1;
        found_index = j;
        success = 1;
        if (found_existing_entry) cache_entry[j] <= 1'b0; // empty line
        break;
      end : if2
    end : for1
    chk_invalidate chk_invalidate_1(.*);
    // checks for invalidates
  end : if1
  checker : chk_invalidate
    ap_invalidate : assert property( ...);
    ap_nothing2invalidate : assert property (..);
  endchecker : chk_invalidate
```

Ben Cohen, VhdlCohen Publishing
Example of a checker application

```vhdl
checker chk_immd(logic the_what,
    string msg,
    logic clk);

// static concurrent assertion
ap_test_now: assert property(@ (posedge clk) the_what) else
    $error("msg, the_what=%b at %t", the_what, $time);
endchecker : chk_immd

// cache_controller
always @ (posedge clk) begin : main_mem_accesses
...
    if (wr) begin : wr_2main // Start of a write
        chk_immd chk_immd_wr_cycle_timer_not_0(
            .the_what(wr_cycle_timer==0),
            .msg("wr signal when memory counter !=0"),
            .clk(clk));
    end
```

Need a static concurrent assertion to enable inline behavior. Immediate assertion is a concurrent assertion.
Potential Use Model of Checkers

- Marks the need of checks during RTL design definition
- Not yet know what is in the checker
  - but feel that there should be something
  - The what’s in the checker can be defined later

Example:

```vhdl
module dut(..);

...

always @ (posedge clk) begin
  if (rd && mode==FAST) begin
    some_code;
    chk_rd_mode_fast  chk_rd_mode_fast_1 (.*);
  end

...
```
Design Experience with Checkers

• **Started with a single checker declared within the module**
  – Because requirements were too loose
  – Single checker became too complex

• **Reconsidered the use model**
  – Smaller targeted checkers
  – As design matured:
    • More checkers created
    • More static concurrent assertions within a checker
    • More elaboration time checks
    • Less clutter in RTL
  – Assertions challenge the design and requirements!
    • Clarify requirements and ease RTL design
      – Even if assertions are correctly or incorrectly written
    • Review of assertions helped the RTL design and debug
Incorrect Assertions yield False Sense of Security

Assertions collocated in checkers help in the review process

```
sequence q_wr_wr;
  int v_addr;
  @(posedge clk) ($rose(wr), v_addr=addr) ##[1:$] wr && addr==v_addr;
endsequence : q_wr_wr

property p_wr_rd;
  int v_addr, v_data;
  disable iff(q_wr_wr_triggered)
  ($rose(wr), v_addr=addr, v_data=wr_data) |->
  ##[1:$] rd && addr==v_addr ##[1:3] rd_data==v_data;
endproperty : p_wr_rd

always @ (posedge clk)
ap_wr_rd : assert property @ (posedge clk) p_wr_rd;
```

```
property p_wr_rd;
  int v_addr, v_data;
  first_match(($rose(wr), v_addr=addr, v_data=wr_data) ##1
  !(wr && addr==v_addr)*1:$] ##1 rd && addr==v_addr) |->
  ##[1:MEM_CYCLES] rd_data==v_data;
endproperty : p_wr_rd
always @ (posedge clk)
ap_wr_rd : assert property(p_wr_rd);
```
Checker / Code Ratio

- **cache_controller** ~ 200 lines with some comments
  - Includes checker instantiations ~ 18 lines
- **cache_mem** ~ 140 lines with some comments
  - Includes checker instantiations ~ 9 lines
- **fifo_mem** ~ 100 lines
  - Includes ~ 100 lines
- **Checkers**
  - chk_rd_cntrl ~ 60 lines
  - chk_invalidate ~ 38 lines
  - chk_immd ~ 7 lines
  - chk_wr_cntrl ~ 90 lines
  - chk_rd_wr_cntrl ~ 30 lines
  - chk_fifo ~ 90 lines
  - chk_reset ~ 7 lines

<table>
<thead>
<tr>
<th>Checkers</th>
<th>LOC</th>
<th>Assertions</th>
<th>Cover property</th>
<th>Declarations</th>
<th>Instantiations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>330</td>
<td>34</td>
<td>6</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DUT</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>440</td>
</tr>
</tbody>
</table>

LOC: checker/DUT ~ 75%
Checker instances/DUT < 3%
Checker Impressions -1
Clear demarcation between RTL and verification code

• Verification and supporting code outside RTL code
  – Reduced clutter in RTL
    Supporting code (signal / functions)
  – No concern about interference or confusion with the RTL
    • Organized and structured solution
    • Amenable to building small to medium verification units

• **Instantiated statically or procedurally within RTL**
  – If instantiated procedurally, then must differentiate
    • static assertions => behave inline
    • procedural assertions => in-block
  – Must be careful in using static concurrent vs procedural concurrent assertions

• **Collocate common assertions statements**
  – Otherwise multiple instantiations will cause duplicate assertions
Checker Impressions -2

- **Facilitates reviews**
  Verification code is collocated

- **Adopts a higher level view of verification**
  Each checker envelops a related aspect of verification

- **Configuration checks at elaboration time are useful**
  Elaboration checks can be included in checkers

- **Shorter code**
  CLK & reset inference

- **Ignored by synthesis tools**
Checker Impressions -3

- Limitations – Not a big issue
  - But ... Must understand the rules

- Limitations on what can be declared
  - What’s not in the body of a checker
    - parameter, localparam and specparam
    - Module, interface, program, class
    - task, void functions, blocking assignments,
    - Functions with side effects
    - if, for, case (in always, initial procedures)
    - hierarchical references

- Limitations on where instantiated
  - Wherever a concurrent assertion may appear
    - Statically or procedurally in always, initial, final
  - Illegal in fork...join, fork...join_any, or fork...join_none blocks
Conclusions --1

• Are checkers useful?
  – Yes, group related code and assertions into single entities
  – Can be instantiated inline with RTL or statically
  – Significantly reduces LOC of RTL
  – Eases review process of verification code

• What was our experience?
  – Process of using checkers:
    • Easy & like it; checkers are cousins of modules
    • Procedural concurrent assertions: useful for in-block behavior when instantiated
      – Different style than normal
    • Static concurrent assertions: useful for inline behavior when instantiated procedurally
  – Tool issues
    • Very few vendors support checkers, as of December 2009
  – Conclusions
    • Checkers represent a good addition to IEEE 1800-2009
    • Checkers should be used in design and verification
Questions?

And, yes, you can stuff many assertions into one checker!

Slides and code can be downloaded from http://SystemVerilog.us/DvCon2010/
Backup slides

• To be used only if questions arise
Checkers Illegal constructs

Those include:

a) Parameter, localparam and specparam
b) Module, interface, program, class
c) Task, void functions, blocking assignments,
d) Functions with side effects
e) if, for, while, case statements (in always, and initial procedures)
f) All hierarchical referencing, into or out of a checker, is disallowed
Checkers Illegal constructs

checker chk_rdhit(
    logic data_vld, busy, rd, wr, cache_exists, clk, reset_n);
always @ (posedge clk) begin : a1
    if(rd) begin : rd1 // Illegal in a checker
        assert(wr == 1'b0);
        if ($past(cache_exists)) // if is illegal in a checker
            assert (busy==1'b0);
    end : rd1
end: a1

function logic rd_cache(logic rd, wr, cache_exists, busy);
    if(rd && !wr && cache_exists & !busy)
        rd_cache = 1'b1;
    else
        rd_cache = 1'b0;
endfunction : rd_cache

ap_rd_hit: assert property (@ (posedge clk)
    rd_noWr(rd, wr, cache_exists, busy));
ap_rd_hit2: assert property (@ (posedge clk)
    rd && cache_exists |-> !wr #!1 !busy);
endchecker : chk_rdhit
Checkers Illegal Instantiations

- It is illegal to instantiate checkers in
  - `fork...join`,
  - `fork...join_any`, or
  - `fork...join_none` blocks
mp_busy_no_rd_wr

wr when busy
Read hit and miss
Good invalidate
Assertion Report with forced failures

Ben Cohen, VhdlCohen Publishing
## Assertion Report with no forced failures

<table>
<thead>
<tr>
<th>Name</th>
<th>Assertion Type</th>
<th>Location</th>
<th>Failure Count</th>
<th>Pass Count</th>
<th>Active Count</th>
<th>Memory</th>
<th>Peak Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cache_controller_tb/ap_mr_rd_memory</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>68</td>
<td>7</td>
<td>1.3K</td>
<td>1.7K</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_fifo_pop_on_miss_full</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>104</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_fifo_push_on_miss</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_invvalidate</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>64B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_miss_then_data_2output</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>0</td>
<td>72B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_read_hit</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>0</td>
<td>128B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_read_miss</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>0</td>
<td>96B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_read_miss_busy_data_vd</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>0</td>
<td>144B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/ap_wr_rd</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>68</td>
<td>7</td>
<td>1.3K</td>
<td>1.7K</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_mem1/cache_mem_handling_if1/ap_noting2invalidate</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_mem1/cache_mem_handling_if1/ap_noting2invalidate_concurrent</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>64B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_mem1/cache_mem_handling/wr_cache_updates/else1_write_if1/ap_no_cache_line_found_4write</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>112</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_mem1/cache_mem_handling/wr_cache_updates/else1_write_if1/ap_no_cache_line_found_4write_concurrent</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>112</td>
<td>0</td>
<td>0</td>
<td>72B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_read_if1/exists/immied_131</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>159</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_read_if1/exists/while1/ap_read_cache_line</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>722</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/cache_read_if1/exists/while1/ap_read_cache_line_concurrent</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>722</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/LRU_fifo/aIllegal_fifo_end_1</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/LRU_fifo/ap_pop_error_1</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>1163</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/LRU_fifo/ap_pop_data</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>158</td>
<td>0</td>
<td>0</td>
<td>72B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/LRU_fifo/ap_push_error_1</td>
<td>Concurrent</td>
<td>SYAon</td>
<td>0</td>
<td>1163</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/main_mem_accesses/rd_from_mem/ap_rd_cycle_timer_not_zero</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/main_mem_accesses/wr_2main/ap_mr_cycle_timer_not_0</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/main_mem_accesses/wr_2main/ap_mr_cycle_timer_not_0_concurrent</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/main_mem_accesses/wr_2main/ap_mr_cycle_timer_not_0</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>845</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/main_mem_accesses/wr_2main/ap_mr_cycle_timer_not_0_concurrent</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>1163</td>
<td>0</td>
<td>0</td>
<td>48B</td>
</tr>
<tr>
<td>/cache_controller_tb/cache_controller_1/host_busy_no_rd_wr</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/cache_controller_tb/test/end_4wr/s0</td>
<td>Immediate</td>
<td>SYAon</td>
<td>0</td>
<td>1050</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>