Hewlett Packard
Enterprise

27th Jan 2016

Formal for Designers

Agile Test Driven Development for ASIC

Jon Buckingham

Agenda

- Agile Connection
- Formal in the design process
- Property organisation
- Simulation Benefits
- Incremental development

Agile Connection

Agile Manifesto

- **Individuals and interactions over Processes and tools**
  - Co-locate teams (inc sw/fw, verification).
  - bugzilla, git etc are useful band aids to global development, but don’t solve this problem
- **Working software over Comprehensive documentation**
  - Release RTL functionality using Test Driven Development (TDD) regularly.
  - Documentation is part of each release.
- **Customer collaboration over Contract negotiation**
  - Review specs and properties with customers
    - software engineers, interacting hardware designers, architects
- **Responding to change over Following a plan**
  - Trickiest for ASIC
  - Test Driven Development (TDD) and incremental development helps
  - Documentation changes
  - The up front plan is still essential!
  - Good communication processes between stakeholders

Agile Connection

Test Driven Development (TDD)

- **Unit Tests**
  - Write unit tests (properties), then code the unit
  - “units” are sub blocks within the classic block
  - done by the designer
  - written as part of each incremental release
- **Acceptance tests**
  - block level simulations
  - constrained random
  - done by verification engineer
- **Integration tests**
  - ASIC level simulations
  - constrained random (usually)
  - done by verification engineer

Agenda

- Agile Connection
- **Formal in the design process**
- Property organisation
- Simulation Benefits
- Incremental development

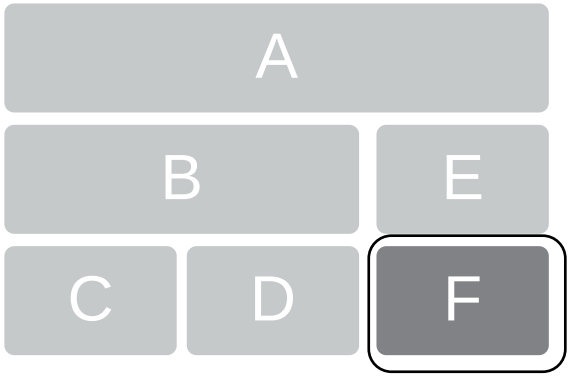
Formal in the Design Process

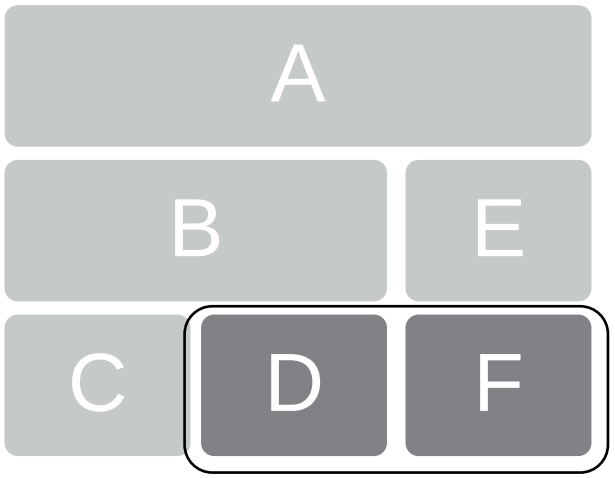
For each sub block

- write properties
  - block initiated operations (aka “master”)
  - outside world initiated ops (aka “slave”)
  - all named master_* or slave_*
- Turn slave ops into assumes with `assume -from_assert`
- Write scoreboards perhaps
- CDC blocks use `abstract -cdc -inject`
- As other sub blocks are added, increase scope of properties
  - only create assumes on the “outside”

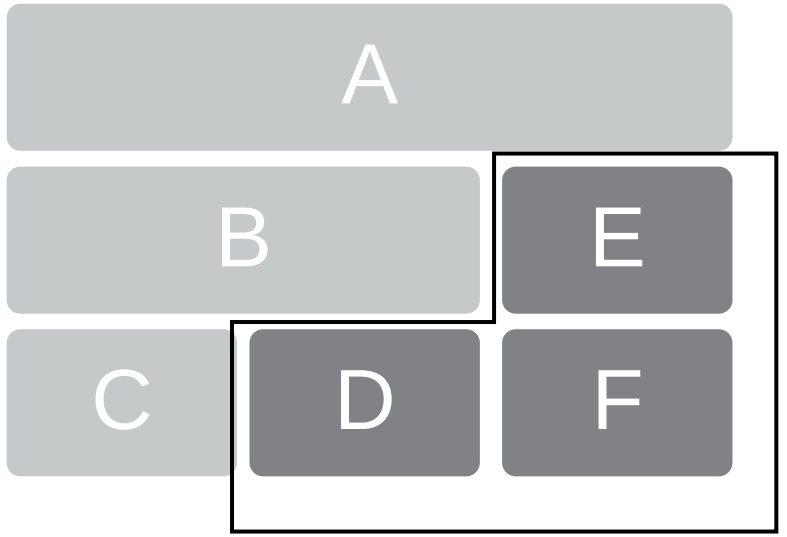
Formal in the Design Process

– Constraints boundary increases with incremental releases





Boundary of “assumes” moves out as sub blocks are implemented. This is controlled by TCL and suitable regexp and hierarchy identification.



Formal in the Design Process

Write Properties as *asserts*: convert to *assume* as necessary

```
// -------------- PROPERTIES ----------------------------
default clocking system_clock @(posedge sys_clk);
endclocking // system_clock
default disable iff (sys_rst_n !== 1'b1);

// -------------- MASTER Asserts ----------------------
csr_master_reset_a          : assert property (first_cycle                              |-> ##[0:1] !cmd.valid);
csr_master_one_read_a       : assert property (cmd.valid && (cmd.payload.op == CSR_RD) |-> outstanding_reads == 0);
csr_master_ready_off_a      : assert property (!cmd_ready                               |-> ##2 !cmd.valid);

// -------------- SLAVE Asserts -----------------------
csr_slave_data_overrun_a    : assert property (outstanding_reads >= 0);
`ifdef JASPERGOLD
   csr_slave_response_time_a : assert property (rd_cmd_in      |-> ##[1:MAX_READ_LATENCY]rdata.valid) ;
   csr_slave_ready_a         : assert property (!cmd_ready     |-> ##[1:$] cmd_ready);
`endif
csr_slave_tag_a             : assert property (rdata.valid    |-> ctag_seen);

// -------------- COVER Properties ----------------------
csr_command_c      : cover property (cmd_in);
csr_wr_c           : cover property (wr_cmd_in);
csr_rdata_c        : cover property (rdata.valid);
csr_wr_within_rd_c : cover property (rd_cmd_in ##1 wr_cmd_in);
```

Then in TCL…

```
#  - this is a CSR slave, so make master asserts assumes
eval assume -from_assert ".*\.csr_master_.*_a$" -regexp
```

Agenda

- Agile Connection
- Formal in the design process
- **Property organisation**
- Simulation Benefits
- Incremental development

Property Organisation

- **Properties at end of RTL files**
- **Properties in dedicated modules**
  - use bind
  - particularly useful for reuse with common interfaces
- **Properties within processes**
  - most useful for state machines
  - don’t have to recreate the state conditions in helper code

Property Organisation

- properties at end of files

```
module rpc_tx_rra
  (
  …
//-----------------------------------------------------
`ifndef SYNTHESIS
//-----------------------------------------------------
   //=====================================================
   // Properties
   //=====================================================
   default clocking system_clock @(posedge rpp_clk);
   endclocking // system_clock
   default disable iff (rpp_rst_n !== 1'b1);

   no_send_while_busy_a: assert property
        ( csr_wr_rpc_rra_send_command |-> !rpc_rra_send_status_r.Busy );
`endif
endmodule // rpc_tx_rra
```

Property Organisation

- properties in dedicated modules

```
bind rpc rpc_pcie_sii_constraints_props
   u_rpc_pcie_sii_constraints_props(.*);

//------------------------------------------------------------------------
module rpc_pcie_sii_constraints_props
//------------------------------------------------------------------------
  (
   input                              rpp_clk,
   input                              rpp_rst_n,
   input  rpp_sii_pkg::rpp_sii_master_t                rpp_sii_master,
   input  rpp_sii_pkg::rpp_sii_slave_t                 rpp_sii_slave,
   input  rpp_pkg::rpp_error_t                         rpp_error
   );
   import rpp_sii_pkg::* ;
   import rpp_pkg::* ;
   // ============== PROPERTIES ============================
   default clocking system_clock @(posedge rpp_clk);
   endclocking // system_clock
   default disable iff (rpp_rst_n !== 1'b1);

   // Check all error signals are single cycle pulse
   pulse_error_a : assert property ( !(rpp_error.part_diag_status
                              & $past(rpp_error.part_diag_status)) );
   endmodule // rpc_pcie_sii_constraints_props
```

Use a separate bind file for property modules that are used in many places, e.g. for interfaces.

Property Organisation

- properties in processes

Minimise this, since it can make RTL harder to read.

```
//-----------------------------------------------------
  always@(posedge rpp_clk or negedge rpp_rst_n) begin : rpp_if_p
  //-----------------------------------------------------
        …
        //-----------------------------------------------
        unique case (state)
        //-----------------------------------------------
          IDLE                : begin
          //-----------------------------------------------
             if (fifo_deq) begin
                new_tlp_actions;
                //-----------------------------------------
                `ifndef SYNTHESIS
                   new_tlp_with_cmd_idle_a: assert property(fifo_cmd);
                `endif
                //-----------------------------------------
             end
          end
          //-----------------------------------------------
          SENDING             : begin
          //-----------------------------------------------
```

Agenda

- Agile Connection
- Formal in the design process
- Property organisation
- **Simulation Benefits**
- Incremental development

Simulation Benefits

- **Properties are all there for simulation**
  - except Jaspergold scoreboards
- **Massively speeds up defect tracking**
  - Many defects are highlighted by failing assert before any simulation checks
- **IP Integration Ambiguity**
  - Capture assumptions about documentation ambiguity as properties
  - Properties will fail during simulation if assumptions are wrong
- **Helps testbench development**
  - design assumptions are captured by properties
  - verification engineers often ask why assert has failed*: it's a test defect!*
  - software usage properties
    - constrain jasper to system usage
    - check verification engineers are using the DUT correctly
- **Do it as you go along**
  - it's much easier than trying to revisit later
  - often more comprehensive
  - 350 properties written, 1300 in simulation due to loops ("typical" ~70k FF block)

Even data transformation blocks (e.g. compression) can benefit from sub block properties

Agenda

- Agile Connection
- Formal in the design process
- Property organisation
- Simulation Benefits
- **Incremental development**

Incremental Development

From Agile Manifesto

- Do all checks continuously with every “micro release”.
- At each release do
  - regression test of all assertions
    - designers run their own unit tests!
  - synthesise
  - rtl-gates equivalence
  - lint
  - up to date documentation
    - spec
    - design (block diag at least, some times internal paths, timing etc)
    - see *agile modelling* (`https://en.wikipedia.org/wiki/Agile_modeling#Documentation`)
      - document continuously and late (aka “not all up front”).
  - doc review, design review, code review
- `http://www.agilesoc.com/articles/agile-transformation-in-functional-verification-part-i/#Incremental_Development`

Summary

Formal and Agile can help *design* it right

- **Agile processes can be applied to ASIC RTL development**
  - Often already doing it
- **Test Driven Development works**
  - Use properties as unit tests
  - Prove using formal tool: requires minimal “testbench” infrastructure
- **Simulation Benefits**
  - Properties are there from the start
  - Accelerate testbench development
- **Incremental Development works**
  - Deliver working features incrementally to verification teams
  - Tie it in with synthesis, lint, CDC checks etc
  - Update documentation incrementally