Guidelines for the “LAB 9: Sequence equivalence checks - RTL to RTL - manual setup” practice sessions

1 Goal of this session

1) ~~Initial introduction to the Cadence Jasper tool~~
2) ~~RTL debug in a formal verification environment~~
3) ~~RTL development in a formal verification environment~~
4) ~~Properties debug in a formal verification environment~~
5) ~~Properties development in a formal verification environment~~
6) RTL to RTL sequence equivalence checking
7) ~~C-to-RTL transactions equivalence checking~~
8) ~~Code quality analysis~~
9) ~~Code reachability analysis~~

2 Step 1 – Download materials

Download the package pointed by the professor, unzip the package and copy that to any location from which the which the Cadence Jasper tool is available.

3 Step 2 – Familiarize with the design specification

Familiarize with the design specification of the block to be used during this practice session. That specification can be found in the package you copied, in the 'documentation/ design_specification.pdf'.

It's the same block, which was already used during dome of the previous sessions.

4 Step 3 – Familiarize with how the SEC flow is structured

Familiarize with the SEC flow setup files, which can be found in the 'scripts' folder.
Also, have a look at the SEC wrapper located in the 'design/dut_toplevel__sec_wrapper.sv' file.

5 Step 4 – Run the SEC flow

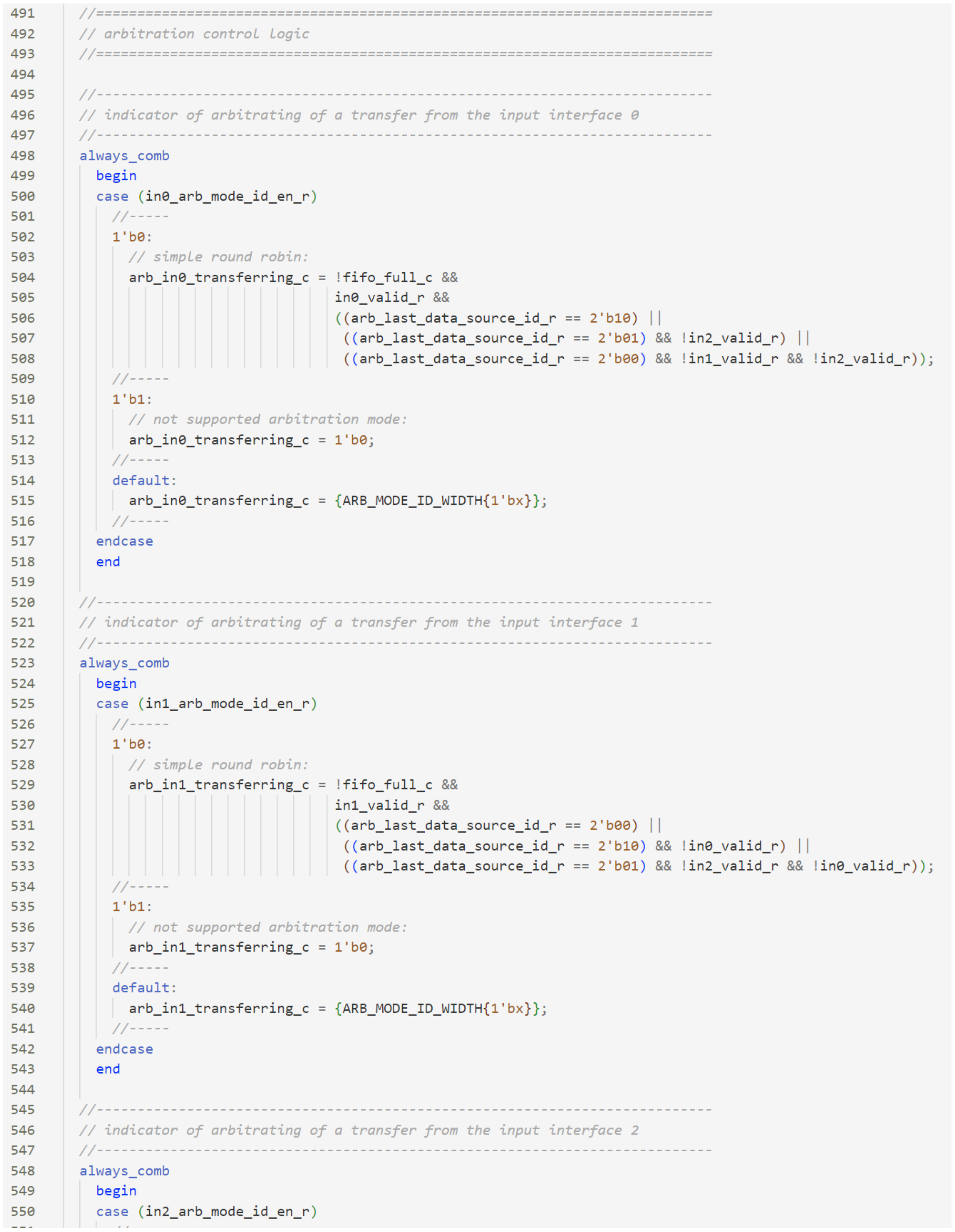
Try to run the SEC flow on the RTL files, which are currently in the 'design' directory. Both the 'spec' and 'imp' files are identical. So, you should have all assertions passing.

To run the SEC flow, use following commands:

- *Optional:* `source <jg_config_script>`
- `cd <unziped_directory>/scripts`
- `jg dut_toplevel_sec_flow.tcl`

6 Step 5 – change arbitration logic

Try to rewrite the arbitration logic (visible in Figure 1) to not use ‘case’ statements. Then, use the SEC flow to check that you’ve not broken anything. Please modify the ‘design/ dut_toplevel__imp.sv’ file.



```
//==========================================================================
// arbitration control logic
//==========================================================================

//--------------------------------------------------------------------------
// indicator of arbitrating of a transfer from the input interface 0
//--------------------------------------------------------------------------
always_comb
  begin
  case (in0_arb_mode_id_en_r)
    //-----
    1'b0:
      // simple round robin:
      arb_in0_transferring_c = !fifo_full_c &&
                               in0_valid_r &&
                               ((arb_last_data_source_id_r == 2'b10) ||
                                ((arb_last_data_source_id_r == 2'b01) && !in2_valid_r) ||
                                ((arb_last_data_source_id_r == 2'b00) && !in1_valid_r && !in2_valid_r));
    //-----
    1'b1:
      // not supported arbitration mode:
      arb_in0_transferring_c = 1'b0;
    //-----
    default:
      arb_in0_transferring_c = {ARB_MODE_ID_WIDTH{1'bx}};
    //-----
  endcase
  end

//--------------------------------------------------------------------------
// indicator of arbitrating of a transfer from the input interface 1
//--------------------------------------------------------------------------
always_comb
  begin
  case (in1_arb_mode_id_en_r)
    //-----
    1'b0:
      // simple round robin:
      arb_in1_transferring_c = !fifo_full_c &&
                               in1_valid_r &&
                               ((arb_last_data_source_id_r == 2'b00) ||
                                ((arb_last_data_source_id_r == 2'b10) && !in0_valid_r) ||
                                ((arb_last_data_source_id_r == 2'b01) && !in2_valid_r && !in0_valid_r));
    //-----
    1'b1:
      // not supported arbitration mode:
      arb_in1_transferring_c = 1'b0;
    //-----
    default:
      arb_in1_transferring_c = {ARB_MODE_ID_WIDTH{1'bx}};
    //-----
  endcase
  end

//--------------------------------------------------------------------------
// indicator of arbitrating of a transfer from the input interface 2
//--------------------------------------------------------------------------
always_comb
  begin
  case (in2_arb_mode_id_en_r)
```

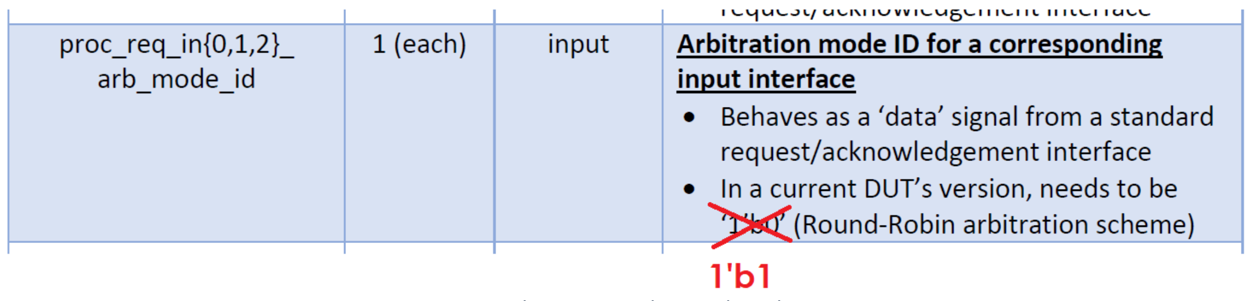
Figure 1. Arbitration logic to be recoded

If you want to skip that step, you can find an example solution here:
`<unziped_directory>/example_solutions/`
`design_with_arbiter_as_non_case_assignment.sv`

If you coded that properly and you don't see any failing assertions, you can introduce some errors to check if the flow catches them. You can find an example with bugs here:
`<unziped_directory>/example_solutions/`
`design_with_arbiter_as_non_case_assignment_with_bugs.sv`

7 Step 6 – change encoding of the arbitration mode

Imagine that your architect changes encoding of the arbitration modes. Now the Round-Robin mode will be indicated as 1'b1 instead of 1'b0 (as visible in Figure 2). Change that in the RTL and use the SEC flow to check that you've done that correctly, in all places. Note that this change may also require some 'hacks' in the SEC wrapper (in the 'design/ dut_toplevel__sec_wrapper.sv' file).



proc_req_in{0,1,2}_ arb_mode_id	1 (each)	input	**Arbitration mode ID for a corresponding input interface** • Behaves as a 'data' signal from a standard request/acknowledgement interface • In a current DUT's version, needs to be ~~'1'b0'~~ (Round-Robin arbitration scheme) 1'b1

Figure 2. Arbitration mode encoding changes

If you want to skip that step, you can find an example solution here:
`<unziped_directory>/example_solutions/`
`design_with_different_arb_mode_coding.sv`

If you coded that properly and you don't see any failing assertions, you can introduce some errors to check if the flow catches them. You can find an example with a bug here:
`<unziped_directory>/example_solutions/`
`design_with_different_arb_mode_coding_with_bug.sv`

8 Step 7 – remove input interface 2

Imagine that your block does not need to support three input interfaces anymore. That is why, please remove input interface 2. It means – remove all ports and logic related to the interface, as shown in Figure 3. Then, use the SEC flow to check that you've done that correctly (that your two-interface block works the same way as the three-interface block with a disabled interface 2).

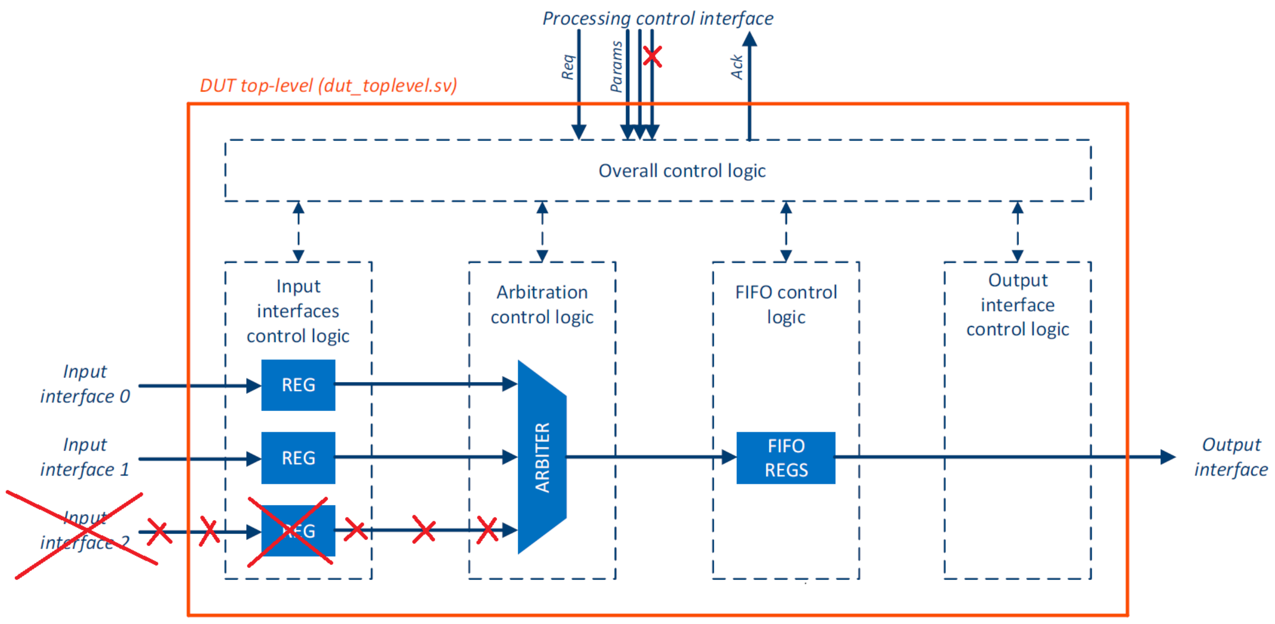

Processing control interface
Req
Params
Ack
DUT top-level (dut_toplevel.sv)
Overall control logic
Input interfaces control logic
Arbitration control logic
FIFO control logic
Output interface control logic
Input interface 0
Input interface 1
Input interface 2
REG
REG
REG
ARBITER
FIFO REGS
Output interface

Figure 3. Removed input interface 2

If you want to skip that step, you can find an example solution here:
`<unziped_directory>/example_solutions/`
`design_with_removed_in2_interface.sv`

If you coded that properly and you don't see any failing assertions, you can introduce some errors to check if the flow catches them. You can find an example with a bug here:
`<unziped_directory>/example_solutions/`
`design_with_removed_in2_interface_with_bug.sv`

9 Step 8 – Play with that formal flow

- Play with the flow
- Introduce some bugs in the RTL and observe if it is caught by the SEC flow

Thank you !