Property Coverage Checker

A brief description of the tool for the evaluation of the model checking process

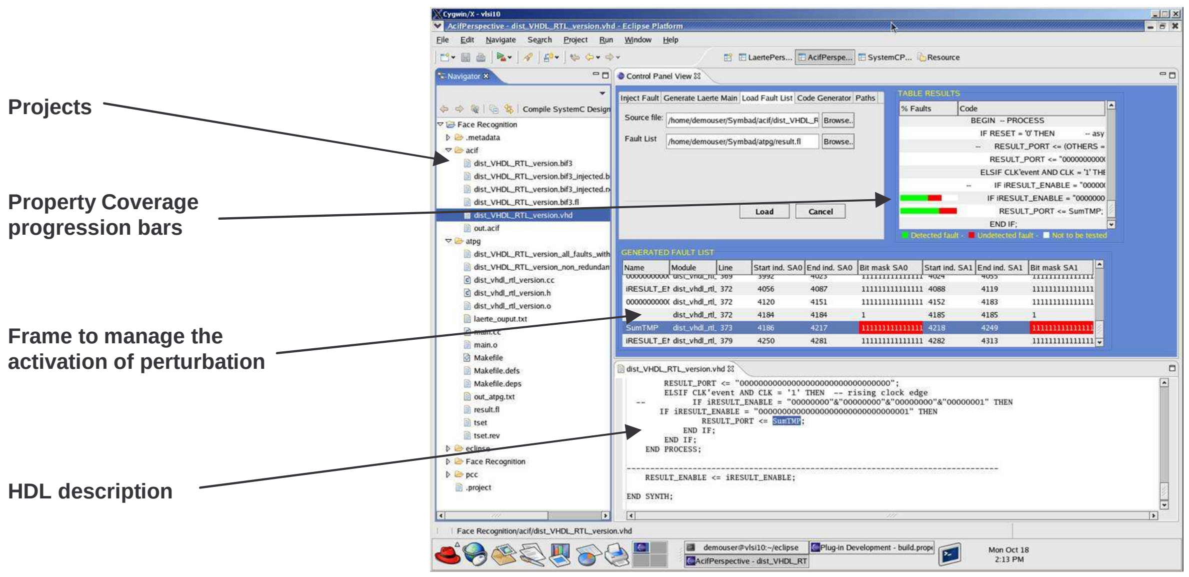
Lack of exhaustiveness is one of the biggest problems related to the use of simulation based techniques for digital system validation. Formal verification aims at overcoming this problem by proving properties on rigorous mathematical models representing the system implementation. In particular, identification of design errors is addressed by means of model checking tools. Thus, formal properties can be defined in order to formally check the correctness of a design implementation with respect to the specification. The defined properties describe the expected behavior of the design, i.e. they represent the golden model for the implementation. However, this golden model could be incomplete, with respect to the informal specification of the design, since some requirements have been only partially formalized. In this case, the implementation could be wrong, even if it fulfils all defined properties.
The Property Coverage Checker (PCC) allows to measure the quality of such golden model to highlight a possible incompleteness of properties. It exploits a high level fault model (bit coverage) and a fault simulation-based approach which correlates property completeness to fault coverage. In this context, bit coverage is used for investigating the capability of properties to identify functional perturbations (faults) of the design implementation. Thus, the degree of property incompleteness is estimated with respect to the *property coverage*, i.e., the percentage of perturbed implementations that properties can detect. (A property detects a perturbation if it fails on the perturbed model, and it holds on the fault-free model.) In this way, a property coverage lower than 100% is consider a symptom of property incompleteness.

PCC workflow

1. Generation of perturbed implementations.
Perturbations of the design implementation are generated by automatically injecting bit coverage faults into its high-level description. Then, a high-level automatic test pattern generator is adopted to identify the set of detectable perturbations (fault). If a perturbation cannot be detected it is discarded, since it cannot provide useful information according to the previously proposed definition of property coverage.
2. Property Coverage Computation.
The property coverage is computed by using a mix of fault simulation and model checking. We have theoretically shown that if a witness of a property detects a fault, then also the corresponding property detects it. Thus, witnesses of properties included in the golden model are extracted and provided as input sequences for fault simulation. Then, faults not detected by witnesses are further analyzed by exploiting a model checking engine, which provides an exhaustive answer about the capability of the golden model to detect such faults.
3. **Incompleteness Evaluation.**
Uncovered faults are graphically highlighted in the HDL design implementation. Thus, by analyzing these uncovered perturbations, the designers can obtain useful information to define missing properties.

Supported formats

The golden model can be defined by using Computation Tree Logic (CTL) or Linear Time Logic (LTL).
The design implementation can be defined by using VHDL or SystemC at RL-level or higher.


Projects
Property Coverage progression bars
Frame to manage the activation of perturbation
HDL description
AcifPerspective - dist_VHDL_RTL_version.vhd - Eclipse Platform
Navigator
Control Panel View
Inject Fault
Generate Laerte Main
Load Fault List
Code Generator
Paths
Load
Cancel
TABLE RESULTS
🟩 Detected fault - 🟥 Undetected fault - ⬜ Not to be tested
GENERATED FAULT LIST

Source file: /home/demouser/Symbad/acif/dist_VHDL_R
Fault List: /home/demouser/Symbad/atpg/result.fl

% Faults	Code
	BEGIN -- PROCESS
	IF RESET = '0' THEN -- asy
	-- RESULT_PORT <= (OTHERS =
	RESULT_PORT <= "0000000000
	ELSIF CLK'event AND CLK = '1' THE
	-- IF iRESULT_ENABLE = "00000
🟩🟥	IF iRESULT_ENABLE = "0000000
🟩🟥	RESULT_PORT <= SumTMP;
	END IF;

Name	Module	Line	Start ind. SA0	End ind. SA0	Bit mask SA0	Start ind. SA1	End ind. SA1	Bit mask SA1
0000000000	dist_vhdl_rtl_	369	3992	4023	111111111111111	4024	4055	111111111111111
iRESULT_EN	dist_vhdl_rtl_	372	4056	4087	111111111111111	4088	4119	111111111111111
0000000000	dist_vhdl_rtl_	372	4120	4151	111111111111111	4152	4183	111111111111111
	dist_vhdl_rtl_	372	4184	4184	1	4185	4185	1
SumTMP	dist_vhdl_rtl_	373	4186	4217	111111111111111	4218	4249	111111111111111
iRESULT_EN	dist_vhdl_rtl_	379	4250	4281	111111111111111	4282	4313	111111111111111

dist_VHDL_RTL_version.vhd

```
        RESULT_PORT <= "00000000000000000000000000000000";
        ELSIF CLK'event AND CLK = '1' THEN  -- rising clock edge
  --          IF iRESULT_ENABLE = "00000000"&"00000000"&"00000000"&"00000001" THEN
       IF iRESULT_ENABLE = "00000000000000000000000000000001" THEN
                RESULT_PORT <= SumTMP;
            END IF;
        END IF;
    END PROCESS;

--------------------------------------------------------------------------------
    RESULT_ENABLE <= iRESULT_ENABLE;

END SYNTH;
```