STAR-DUST: Hierarchical Test of Embedded Processors by Self-Test Programs

Peter Marwedel[1], Ulrich Bieker[2], Martin Kaibel[3], Walter Geisselhardt[4]

[1] Dept. of Computer Science XII, University of Dortmund, marwedel@acm.org

[2] PRO DV, Dortmund, [3] Siemens, München, [4] Dept. of Electr. Eng., University of Duisburg

Abstract: This paper describes the hierarchical test-generation method STAR-DUST, using self-test program generator RESTART, test pattern generator DUST, fault simulator FAUST and SYNOPSYS logic synthesis tools. RESTART aims at supporting self-test of embedded processors. Its integration into the STAR-DUST environment allows test program generation for realistic fault assumptions and provides, for the first time, experimental data on the fault coverage that can be obtained for full processor models. Experimental data shows that fault masking is not a problem even though the considered processor has to perform result comparison and arithmetic operations in the same ALU.

1 Structure of STAR-DUST

The tools and data formats currently used for STAR-DUST are shown in fig. 1.

We start with an RT-level structural description of the processor under test. Input formats currently available for this purpose include VHDL or MIMOLA.

From this specification, we have to obtain equivalent gate level descriptions. For this purpose, we run the SYNOPSYS logic synthesis tools for each of the RT-level components. In fig. 1 it is assumed that a processor `SIMPLECPU` consists of just four components: register file `reg_file`, an ALU, an ALU multiplexer, and a controller.

Next, we select a heuristic order for processing RT-level components (in figure 1, components drawn in front are processed first). Respecting that order, we do the following for each RT-level component:

- We use the DUisburg Sequential Test generator DUST [GK91] for obtaining test pattern sets for the current component (e.g. the register file). Test patterns produced by DUST have to be translated into the test code language (TCL) accepted by the next tool.

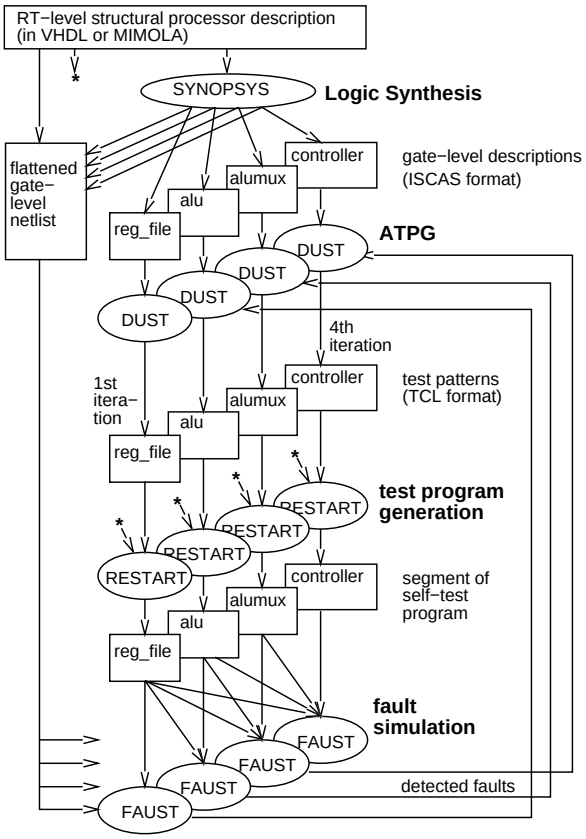

Figure 1: STAR-DUST test generation process

- Self-test programs are synthesized using RESTART. RESTART produces programs justifying test patterns and evaluating test responses.

  Example: Consider a test pattern for an ALU. Suppose control code "10" activates the add operation of the alu, and assume that binary values "0111" and "0001" have been selected as test patterns by DUST. Then, a test for the add operation is specified in TCL by the following state-

RT-component	gates	DFF's	flt's	det	untest	f_untest	aborted	fault coverage	efficiency
reg_file	316	64	2256	2252	0	4	0	99.82%	100.00%
alu	70	0	436	436	0	0	0	100.00%	100.00%
alumux	13	0	76	76	0	0	0	100.00%	100.00%
controller	68	8	488	480	6	2	0	98.36%	100.00%
SIMPLECPU	467	72	3256	3244	6	6	0	99.63%	100.00%

Table 1: SIMPLECPU results

ment:

TEST alu(0111,0001,10);

The result should be 1000. RESTART generates code, which activates the + operation and checks the result by a conditional jump:

```
IF 0111 + 0001 = 1000
THEN increment program counter
ELSE jump to error label;
```

Details on the code generation technique can be found in [BM95, Bie95].

- Programs generated by RESTART are then used as initial stimuli for fault simulation at the gate-level. This way, the coverage of the programs produced by RESTART can be computed. Fault simulation is based on the gate-level stuck-at fault model. For fault simulation, we use FAUST (FAult Simulation Tool), a very efficient single pattern, single fault propagation method. Information about covered faults is exploited in the next cycle of the loop in order to reduce the size of the required test pattern set.

The loop terminates if all RT-level components have been considered.

2 Features of STAR-DUST

STAR-DUST meets a number of objectives:

- STAR-DUST is the first test generation process computing the fault coverage which can be achieved by self-test programs.
- STAR-DUST is a hierarchical test generation process reducing the complexity of generating test patterns for the entire processor to that of generating test patterns for each of the components. Test generation proceeds at a high level of abstraction (using RESTART) while at the same time preserving the high fault coverage through gate level fault modelling.
- STAR-DUST reduces the length of the test program by considering faults covered by tests generated for one component during subsequent generation of test patterns.
- Fault simulation is effectively used for validating the consistency of RT-level and gate-level models.
- Processor testability can be improved with additional redesign cycles.

3 Results for SIMPLECPU

The entire self-test program for SIMPLECPU consists of 250 instructions. Table 1 gives information and results concerning the example processor SIMPLECPU.

For every RT component and the entire processor, table 1 shows the number of gates, the number of D-flip-flops, the number of stuck-at faults, the number of detected faults, the number of untestable faults, the number of functional untestable faults, the number of aborted faults, the fault coverage for the stuck-at fault model, and the efficiency.

References

[Bie95] U. Bieker. Retargetable compilation of self-test programs using constraint logic programming. *in: P. Marwedel, G. Goossens (ed.): Code Generation for Embedded Processors, Kluwer Academic Publishers*, 1995.

[BM95] U. Bieker and P. Marwedel. Retargetable self-test program generation using constraint logic programming. *32nd Design Automation Conference*, 1995.

[GK91] N. Gouders and R. Kaibel. Advanced techniques for sequential test generation. *Proc. ETC*, pages 293–300, 1991.