Partition-Based Faults Diagnosis of a VLIW Processor

Davide Sabena(✉), Matteo Sonza Reorda, and Luca Sterpone

Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy
{davide.sabena,matteo.sonzareorda,luca.sterpone}@polito.it

Abstract. Reconfigurable systems are increasingly used in different domains, due to the advantages they offer in terms of flexibility: reconfigurability can also be used for managing possible faults affecting a circuit, when fault tolerance is the target. In this case the system must be able to (1) detect any possible fault, (2) identify the module (or partition) including it, and (3) take proper actions able to overcome the problem (e.g., by substituting the faulty module with a spare one). In this chapter, we address the point (2) when a Very Long Instruction Word (VLIW) processor is used by resorting to a Software-Based Self-Test (SBST) approach. SBST techniques have shown to represent an effective solution for permanent fault detection and diagnosis, both at the end of the production process, and during the operational phase. When VLIW processors are addressed, SBST techniques can effectively exploit the parallelism intrinsic in these architectures. In this chapter, we propose a new approach that starting from existing detection-oriented programs generates a diagnosis-oriented test program. Moreover, we propose (1) a detailed analysis of the generated equivalence classes and (2) a solution aimed to maximize the diagnosability of the modules composing the VLIW processor under test, thus perfectly suiting the needs of reconfigurable systems. Experimental results gathered on a case study VLIW processor show the effectiveness of the proposed approach: at the end of the presented method, the faulty module is always identified.

Keywords: Software-based diagnosis · Partition-based diagnosis · VLIW processor

1 Introduction

Reconfigurable processors [1] are increasingly used in different domains. Their key characteristic lies in the fact that they can be easily configured to match the specific requirements of the target application, e.g., in terms of performance, size, and power consumption, thus possibly making them more convenient than traditional processors. Very Long Instruction Word (VLIW) processors [2] represent a popular choice among reconfigurable processors.

When the system is used for a safety- or mission-critical application, dynamic reconfigurability may be exploited to face the effects of permanent faults: in this case the processor undergoes some test during the operational phase, aiming at detecting possible faults affecting the hardware. The test can be activated either at a specific moment in time (e.g., at power on), or periodically. As soon as a permanent fault is

A. Orailoglu et al. (Eds.): VLSI-SoC 2013, IFIP AICT 461, pp. 208–226, 2015.
DOI: 10.1007/978-3-319-23799-2_10

detected, a diagnostic procedure is activated to identify the faulty partition, so that proper actions can be taken, e.g., by substituting it with a spare one, thus restoring the system integrity.

When adopting this solution, we need an effective test procedure, able to detect the highest percentage of possible faults while matching the requirements of a test performed during the operational phase (e.g., in terms of duration, size, invasiveness). Some previous works in the area [3, 4] showed that when considering VLIW processors, these goals can be achieved resorting to a functional approach, in which a suitable test program is executed and the produced results are observed (*Software-Based Self-Test* or SBST [5]). The SBST test programs can be generated starting from the processor netlist or (with some limitations) from its RT-level description [6]. Some recent work demonstrated that thanks to their regular structure, test program generation can be even automated in the case of VLIW processors [4, 7], thus overcoming the major limitation of the SBST approach, lying in the high cost for manually generating the test. On the other side, the SBST approach shows some advantages with respect to the structural approach (e.g., based on scan) when the in-field test is considered, mainly due to its easier usage and much lower area overhead.

On fault detection, the application code is typically suspended in the faulty system, thus preventing the fault to produce critical misbehaviors. Then, the system activates a diagnostic procedure, whose goal is to identify the faulty partition out of those composing the processor: in this context, each partition represents the minimal unit that can be repaired or substituted if faulty. This procedure can resort once more to SBST, i.e., to the execution of a suitable test program, whose results allow identifying the faulty partition [8].

Since the basic motivation for this work is to support the design of highly dependable systems based on dynamic reconfiguration, the goal of our diagnostic approach is to identify the faulty partition, rather than the specific fault responsible for a given misbehavior, as in other works (e.g., [9]). A similar approach was followed in [10], where the issue of self-adapting the test so that it takes into account possible units, which have been already, labeled as faulty is considered. However, no one of the previous works gives a systematic method to generate diagnosis-oriented test programs, as we do in this chapter.

This chapter proposes a method able to identify the module including the fault affecting a VLIW processor, taking into account the intrinsic features of this particular kind of processors. The proposed method is mainly composed of two parts: initially, we focus on the issue of writing an accurate diagnostic SBST test program for a generic VLIW processor. The proposed approach is based on exploiting an existing test program (targeting to fault detection, only), and on applying a set of techniques for improving it so that it can hold sufficient diagnostic properties [11] with respect to a previously defined partitioning of the processor. In the second part, the method acts on the initial partitioning, optimizing it so that the achieved diagnosability is maximized. Besides this optimization algorithm, with respect to [11] this chapter also presents a detailed analysis of the results obtained through the diagnostic test program, highlighting a number of cases in which the faults belonging to different partitions cannot be distinguished using a software-based solution (i.e., they are functionally equivalent).

The basic idea behind the first part of the method is to exploit the regularity and parallelism characterizing a VLIW processor. In particular, the technique we propose is based on splitting the original test program in small pieces (called *fragments*), and then modifying each fragment in such a way that it performs the same operation using different resources (e.g., different registers, or different ALUs). By checking which ones of the replicas of the original fragment (called *brother fragments*) generates a misbehavior, we can identify the faulty module.

In the last part of the chapter we explain how to further improve the diagnosability of a generic VLIW processor by implementing a clever partitioning. More in particular, since in the dynamic reconfigurability scenario the aim is the identification of the faulty module and not of the single possible fault, analyzing the composition of the equivalence classes generated by the diagnostic program allows to understand that (1) due to the implementation rules of the processor under test, it is not always possible to distinguish the faults in a module from those in other modules, and (2) by slightly modifying the partitioning it is possible to achieve a very high level of diagnosability.

The method we propose has been experimentally evaluated resorting to a sample VLIW processor [12]: initially, an existing test program aimed at fault detection, only, has been modified and improved, thus obtaining a diagnostic test program whose characteristics (in terms of size, duration and diagnostic capabilities) have been evaluated and compared with those of the original test program. Secondly, applying the second part of the method, the diagnosability of the considered processor has been maximized.

The chapter is organized as follows. Section 2 includes some background about the architecture of a VLIW processor. Section 3 provides an overview about diagnosis of circuits and processors and introduces some notation and vocabulary. Section 4 explains the proposed method. Experimental results on the selected case study and their analysis are presented in Sect. 5. In Sect. 6 we explain how to further improve the diagnosability of the considered VLIW processor by acting on the partitioning. Finally, conclusions and future works are described in Sect. 7.

2 VLIW Architecture Summary

VLIW processors are increasingly employed in systems requiring high performance combined with low power consumption. From a hardware point of view, the two most significant differences between a superscalar processor and a VLIW processor are:

- all the operations are executed by parallel independent *Computational Domains* (CDs), each one characterized by its own Functional Units;
- the scheduling adopted by the processor for instruction execution is totally static, since the compiler assigns the execution of each instruction to a determinate CD. Consequently, in a traditional VLIW processor there isn't any hardware scheduler of the operations.

As shown in Fig. 1-a, from a software point of view the VLIW assembly code is composed of a sequence of macro-instructions (also called *Bundles*): each macro-instruction is composed of a sequence of instructions. Each instruction code embeds the

information items required to assign its execution to a specific Computational Domain, which is selected at the compile time.

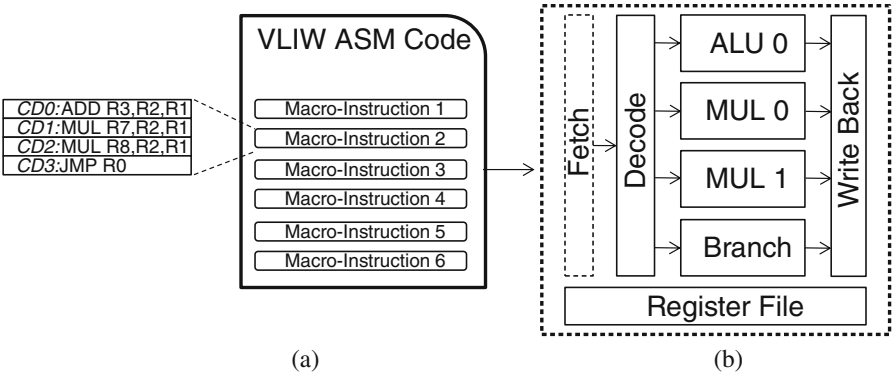

Fig. 1. Example of a VLIW Instruction Code format (a) and of a VLIW Architecture (b)

This scheme proved to be able, in some applications, to significantly reduce the power consumption and the silicon area if compared to traditional superscalar processors. Furthermore, the Instruction Level Parallelism (ILP) can be adequately exploited (at least in the case of data intensive applications), since a good compiler is able to detect which instructions can be executed in parallel by checking the entire program at compile time [2].

As shown in Fig. 1-b, the architecture of a generic VLIW processor is fully parametric, so that different options, such as the number and type of functional units (FUs), the number of multi-ported registers (i.e., the size of the register file), the width of the memory buses and the type of different accessible FUs can be modified depending on the application requirements. The VLIW manifest collects all the characteristics of a specific VLIW processor: it specifies the number of Computational Domains, the number and type of the Functional Units embedded into each of them, the size and access mode of the multiport Register File [4] and any other feature that must be taken in account when developing the code for the considered processor.

Considering the regularity and simplicity of the typical VLIW architecture, this kind of processors is perfectly suited for being adopted in reconfigurable systems [13], either (1) to match variable application constraints and goals, or (2) to implement highly dependable systems [14]. In the first case, the processor is implemented resorting to a programmable device, and the different components are dynamically mapped on the available resources in such a way to optimize the execution of the target application; in the second case, some spare resources are embedded in the architecture, and they are used to replace some faulty module as soon as a permanent fault is detected.

3 Basics on Diagnosis

Let call $F = \{f0, f1, \ldots, fn\text{-}1\}$ the set of n faults that can affect the Unit Under Test (UUT) we are considering. Each of these faults causes the UUT to produce a given output

behavior b (also called *syndrome*) when a given sequence of input stimuli I is applied; let denote by b_i the output behavior produced by fault f_i, and b_g the output behavior of the fault-free circuit. Clearly, $b_i = b_g$ for all undetected faults f_i. When SBST is considered, the assumption is often made, that the output behavior corresponds to the set of values left by the program in memory at the end of its execution. We will make this assumption throughout this chapter. The key rationale behind it is the ease of its implementation in practice, when test (or diagnosis) are run during the operational phase. Therefore, $b_i = b_j$ iff the two faults f_i and f_j produce the same output values in memory at the end of the execution of the test (or diagnosis) program. From a practical point of view, storing a signature of the values produced by each fault may allow to easily identify the existing faults [11]. Alternative solutions avoiding the storage even of this compressed form of fault dictionary can also be considered [15].

A given pair of faults (f_i, f_j) is said to be *distinguished* by a given sequence of input stimuli I iff $b_i \neq b_j$. Otherwise, they are said to be *equivalent wrt I*. All faults that are equivalent wrt to a given sequence of input stimuli I are said to belong to the same *Equivalence Class wrt I*. A detected fault f_i is said to be *fully diagnosed* by a sequence of input stimuli I iff any couple of faults (f_i, f_j) including f_i is distinguished by I. Since two faults f_i, f_j can never be distinguished if they are functionally equivalent, the number of fully diagnosed faults in a circuit is typically rather low [11].

Several possible metrics can be adopted to measure the diagnostic capabilities of a sequence of input stimuli I [16].

When diagnosis is used in a reconfigurable system for identifying the partition including the fault, the precision required is lower than in other situations where diagnosis is required (e.g., for yield ramp-up): in fact, the final goal in this case is to be able to distinguish all pairs of faults belonging to different partitions, while distinguishing pairs of faults belonging to the same partitions is not of interest. Hence, for the purpose of this chapter we will exploit a metric called *Diagnostic Capability*, or *DC(I)*, which corresponds to the percentage of faults belonging to an Equivalence Class wrt I composed of faults all belonging to the same partition. In the ideal case in which DC(I) is 100 %, this would mean that I is able to always identify the partition where the fault is located. We will also exploit the notion of *Fully Diagnosed Fault with respect to Partitions* (*FDP*), which is a fault belonging to an Equivalence Class composed of faults all belonging to the same partition. Clearly, DC(I) is the percentage of FDP faults with respect to the total number of faults.

4 Diagnostic Test Program Generation

In this section we describe a new method that allows to generate diagnostic programs for a generic VLIW processor, once its specific configuration is known.

As shown in Fig. 2, the flow aimed at the generation of the diagnostic program is composed of two main parts, denoted as *classification* and *brother fragment generation*. The result of these two steps is an accurate test program with an improved diagnostic capability.

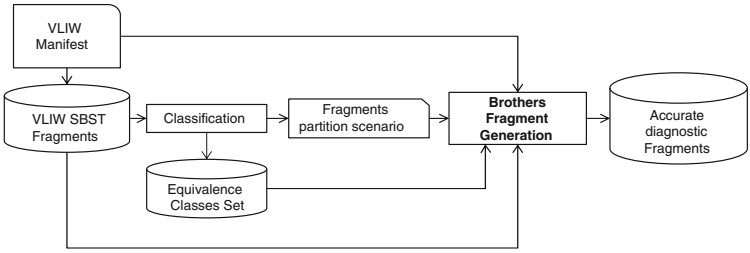

Fig. 2. The flow of the proposed diagnosis method

The proposed flow requires two main inputs. The former is the manifest of the VLIW processor under analysis, which contains all the features of the processor itself (which is supposed to be organized into a few partitions). The latter is a collection of small test programs aimed at fault detection, called *fragments*: each fragment performs a few test instructions (aimed at exciting a specific fault or group of faults) plus some other instructions needed to prepare the required parameters and make the results of the test instruction observable. The fragments have been generated splitting the original SBST programs [4]: the fragments should contain the lowest possible number of instructions and detect the lowest possible number of faults (while still maintaining the same total fault coverage). The set of the initial fragments is called *Initial Test Program*.

4.1 Classification

The classification part aims at computing the Equivalence Classes with respect to the Initial Test Program. This task can be easily performed resorting to commercial Fault Simulation tools and its final result (which requires some further custom post-processing) is the assignment of each fault either to an Equivalence Class composed of faults belonging to a single partition (in which case the fault is labeled as FDP) or to an Equivalence Class including faults belonging to different partitions.

In practice, this phase requires performing the Fault Simulation of each fragment, then processing the data base storing the syndrome of each fault, and finally computing the Equivalence Classes.

The result of this part of the method is the Fragment Partition Scenario, which consists of a database storing for each partition the list of faults belonging to it as well as their syndrome.

4.2 Brother Fragment Generation

The brother fragment generation part is oriented to the generation of new diagnostic fragments capable to improve the overall custom fragment diagnostic capability, thus increasing the DC(I) metric of the addressed VLIW partitions. The flow, illustrated in Fig. 3, is composed of four phases: (1) *analysis of multiple partitions*, (2) *couple faults extraction*, (3) *module identification* and (4) *creation of new fragments*. The 4 phases are repeated until a given stopping condition (e.g., based on maximum computational time, or on the achieved diagnostic capabilities) is reached.

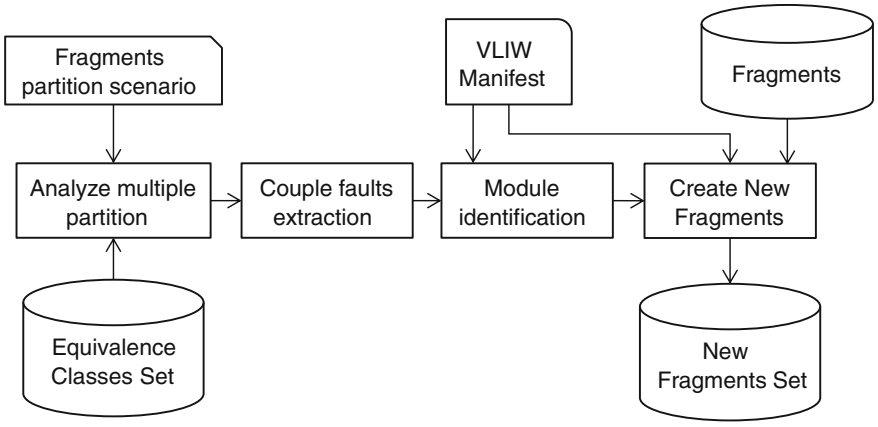

Fig. 3. The flow of the brother fragment generation

The "analyze multiple partition" phase elaborates the fragment partition scenario database comparing equivalence classes including faults belonging to two partitions. In details, in this step, all the equivalence classes are compared and the couple of faults equivalent and belonging to different VLIW partitions are identified.

Once the list of equivalent faults is generated, the "couple faults extraction" phase selects each couple of two fault locations, one belonging to the partition *i* and the other belonging to *j*.

The "module identification" phase identifies the location of the two faults *i* and *j*, analyzing the fault location hierarchy with respect to the VLIW manifest information; the result of this phase is the identification of the VLIW circuit resources involved by each fault.

Finally, the "create new fragments" phase is executed. Basically, this phase elaborates the original test fragments involved into the VLIW resource module identified by the Module identification phase and generates a new set of fragments modifying the resource used by the original test instructions. In this way, the final test program includes two or more different fragments, which are supposed to fail alternatively, depending on whether one or the other of the two partitions we want to distinguish are faulty. The pseudo-code of the Create New Fragments phase is reported in Fig. 4.

The algorithm needs the code of the original test fragment (OF), the VLIW manifest (VM) and the selected rule (R) which is provided by the module identification phase. There are two main rules that can be used for the generation of the new fragments: the first, denoted as R1, is a *register re-allocation rule* and it implies that the brother fragment will contain the same instructions of the original one, but each instruction will use different registers. In this way, by checking the results of the two fragment execution, we are able to understand if the fault is the register file (in case the two fragments results are both wrong) or one of the other VLIW module involved by the two fragments. The second rule, denoted as R2, is a *resource re-allocation rule*: simply, the new brother

fragment will use a different VLIW Functional Unit to execute the test instruction of the fragment.

```
1. OF = Original Fragment;
2. VM = VLIW Manifest;
3. R  = Selected Rule;
4. FL = Faults List;
5. Analysis of the Original Fragment OF - Identification of:
   5.1. A: Test Instruction (TI);
   5.2. B: Instructions that set-up the registers used by TI;
   5.3. C: Instructions that forward  the produced results to
        observable locations;
   5.4. The registers used by A, B and C;
6. Selection of the new resources, according to:
      • the selected rule R;
      • the VLIW Manifest VM;
7. Brother Fragment Generation: Assigns to A, B and C the new
   resources, according to:
      • the selected rule R;
      • the VLIW Manifest VM;
8. FL = Fault Simulation of the Brother Fragment.
```

Fig. 4. The pseudo-code for the "create new fragments" phase

According to OF, VM, and R the algorithm analyzes the original test fragment considering the used test instruction (TI), the VLIW functional unit (FU), the registers used as operands (RI) and the registers used to forward the produced results to observable locations (RO). Finally, it selects a new set of resources and on the basis of the defined rules it generates a new fragment.

In Table 1, an example of original fragment and two corresponding brother fragments is shown; in this example we address a fragment in which the test instruction aims at the adder functional unit embedded in the Computational Domain 0 (referred as CD0). The first brother fragment has been generated with the rule R1 (i.e., the register re-allocation rule), in order to dismember an equivalence class containing faults embedded in the register file and in the adder functional unit of CD0. Consequently, the new brother fragment will be generated changing all the registers used to perform the test instruction and to forward the result in the data memory, without changing the functionality of the original fragment. The second brother fragment, instead, has been generated with the rule R2 (i.e., the resource re-allocation rule): practically, the test instruction of the original fragment has been moved from the computational domain 0 to the computational domain 1, leaving unaltered the other instructions composing the original fragment. In this way, if the results of the two fragments are both wrong, the fault is definitely not embedded in one of the two functional units executing the test instructions, but it belongs to another module used by the two fragments.

Table 1. An example of two brother fragments generated from the same original fragment.

Original Fragment
---< macro-Instruction 1 >--- CD0: mov $r0.1 = 11111…1 CD1: mov $r0.2 = 00000…0 ---< macro-Instruction 2 >--- CD0: add $r0.3 = $r0.1, $r0.2 /*Test instr.*/ CD1: nop ---< macro-Instruction 3 >--- CD0: stw 0[$r0.63] = $r0.3 CD1: nop ----------
1st Brother Fragment – Rule R1
---< macro-Instruction 1 >--- CD0: mov $r0.7 = 11111…1 CD1: mov $r0.8 = 00000…0 ---< macro-Instruction 2 >--- CD0: add $r0.9 = $r0.7, $r0.8 /*Test instr.*/ CD1: nop ---< macro-Instruction 3 >--- CD0: stw 0[$r0.63] = $r0.9 CD1: nop ----------
2st Brother Fragment – Rule R2
---< macro-Instruction 1 >--- CD0: mov $r0.1 = 11111…1 CD1: mov $r0.2 = 00000…0 ---< macro-Instruction 2 >--- CD0: nop CD1: add $r0.3 = $r0.1, $r0.2 /*Test instr.*/ ---< macro-Instruction 3 >--- CD0: stw 0[$r0.63] = $r0.3 CD1: nop ----------

5 Experimental Results

In this section we present the experimental results obtained using the ρ-VEX VLIW [12] processor as a case study. The ρ-VEX processor is a generic and reconfigurable VLIW processor written in VHDL language by researchers of the Delft University of Technology. The ρ-VEX processor includes most of the features of VLIW processors used by industry. For the purpose of this chapter, we considered the stuck-at fault model, although the method can be easily extended to deal with other fault models. In order to perform the stuck-at fault simulation experiments, we synthesized and implemented the ρ-VEX processor using a standard ASIC gate library. The total number of stuck-at faults in the resulting netlist is 335,336.

We divided the ρ-VEX processor in 10 partitions: the fetch unit, the decode unit, the general-purpose register file, the branch-management register file, the write-back unit, and the four Computational Domains in which the functional units are embedded. Clearly, these partitions are not uniform (in terms of number of contained resources).

In Sect. 6.3 we present a method able to create homogeneous partitions without changing the diagnostic results achieved by the method described in the following paragraphs.

Considering the diagnosis goal, in this Section we address only the most relevant partitions of the ρ-VEX processor, i.e., the register file and the four Computational Domains (CD0 to CD3). The number of faults enclosed in each of the four Computational Domains is not exactly the same, since some of the functional units embedded in each of them are different: for example, CD0 includes a branch unit, while CD3 embeds a memory access unit, while all the CDs include an ALU unit.

We also wrote a program (composed of about 1,200 lines of C++ code) able to compare the fault lists generated by the fault simulation step; the goal of this program is to implement the classification phase, i.e., performing the computation of the equivalence classes with respect to the adopted test programs. Our tool also identifies FDP faults, and provides information about the remaining faults.

By referring to the above 5 partitions in the ρ-VEX processor we applied the proposed method and generated the diagnostic test program. As a starting test program we use the set of fragments used for the optimized generation of an SBST program addressing the ρ-VEX processor, generated with the method proposed in [4]; this set is a selection, from an exhaustive set of possible fragments, of the fragments that allow to maximize the stuck-at fault coverage, minimizing the test size and length.

The experimental results we gathered are reported in Table 2, which includes the percentage of FDP faults with respect to the total number of faults of each partition, i.e., the Diagnostic Capability. The first column of Table 2 (denoted as Optimized SBST) is the original test set, composed of 244 fragments; its diagnostic level is rather low for all the considered partitions, since this is optimized in terms of size and length, which are often conflicting goals with respect to diagnosis. The stuck-at fault coverage reached by this test program is 98.2 % with respect to all the resources of the considered VLIW processor.

Table 2. Diagnostic capability

Partition	Method		
	Optimized SBST	Exhaustive fragments set	Proposed approach
Register file	62.82 %	84.23 %	87.17 %
CD0	77.12 %	77.79 %	83.74 %
CD1	80.12 %	81.56 %	88.39 %
CD2	79.99 %	80.34 %	88.23 %
CD3	70.80 %	72.14 %	81.65 %

The first step towards the improvement of the Diagnostic Capability is the use of the whole fragments set generated resorting to the method described in [4]. The results obtained with this approach are shown in the second column of Table 2 (Exhaustive Fragments Set). The improvement of the diagnosis resolution is greater when the register file is considered (the improvement for this partition is more than 21 %), while it is limited for the Computational Domains. This is mainly because the considered set of fragments is composed of 748 fragments, and 68 % of them target the test of a portion of the register file itself.

The final step of the proposed flow is the evaluation of the diagnostic capabilities of an ad-hoc fragments set, composed of the fragments of the Exhaustive Fragments Set with an additional set of fragments brothers, developed with the method proposed in Sect. 4. For the purpose of this chapter, we generated the brother fragments only for the fragments addressing the test of the ALUs (that are the most relevant components of each CD in terms of number of stuck-at faults). Moreover, we developed the brother fragments also for the memory unit (which is embedded in CD3), since this unit is used by all the fragments in order to save the results of the test instructions in the data memory; consequently, there are many equivalence classes containing a fault of this unit and an efficient diagnostic of this module is required. The resulting set of fragments is composed of 1,056 fragments, of which 308 are brother fragments. The CPU generation time for the brother fragments was approximately 21 h, of which about 85 % used for the fault simulation; the computational time has been evaluated on a workstation with an Intel Xeon Processor E5450. As shown in Table 2, the improvements due to this approach are evident if the partitions containing the ALUs (referred as CD1, CD2, CD3 and CD4) are considered: the capability to recognize if a fault is enclosed in one of these partitions is improved of about 8 % with respect to the previous approaches. The resulting diagnosability is not uniform for all the four CDs since, as explained previously, the functional units embedded into these partitions are not the same.

We also made an analysis about the Equivalence Classes wrt the last test set, focusing on those that include faults belonging to more than one partition (i.e., neglecting all FDP faults). Analyzing these equivalence classes, only, it is possible to notice that about 95 % of them are classes only including faults belonging to the same partition; moreover, if the remaining classes are considered, about 60 % of them are equivalence classes enclosing faults belonging to 2 partitions, while about 35 % are classes enclosing faults belonging to 3 different partitions, as shown in the graph of Fig. 5. The above results show that even when the diagnostic resolution of our method is not enough to identify the single partition including a fault, still it is able to identify the couple of “candidate” partitions in about 60 % of the cases.

Finally, in Table 3 some more information about the size and the execution time of the final fragments set are shown. These results confirm that optimizations, in terms of size and length, are often conflicting goals with respect to diagnosis.

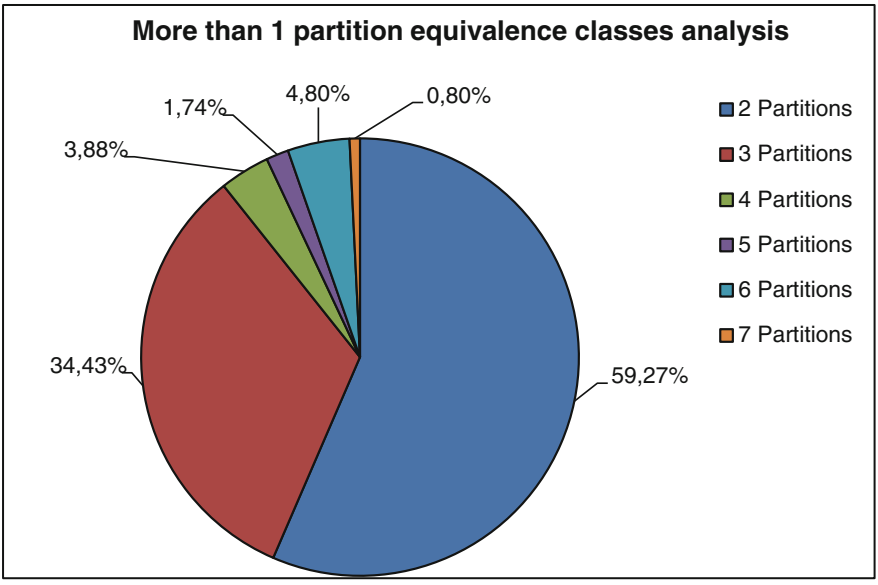

Fig. 5. Analysis of Equivalence Classes including faults belonging to more than one partition

Table 3. Size and duration of the different test sets

Method	Size [KBs]	Execution time [Clock Cycles]
Optimized SBST	1,926	10,601
Exhaustive fragments set	3,429	17,049
Proposed approach	4,899	24,356

6 Equivalence Classes Analysis

In this section, we present a detailed analysis aimed at (1) better understanding the achieved results, i.e., identifying the reasons that prevent the diagnostic metrics to be further increased, and (2) understanding how it is possible to reach a complete diagnosability of the partitions composing the addressed VLIW processor. Finally, we present an equivalence class-based technique aimed at improving the partitioning of the processor resources in order to achieve a scenario in which all the partitions are composed of a comparable number of logic resources, in order to make the proposed method suitable to be used in a dynamic partial reconfiguration environment [13].

6.1 VLIW Equivalence Classes Analysis

Using the Software-Based diagnosis method described in the previous sections, it is possible reach a high level of diagnosability. However, due to the hardware structure of

the considered VLIW processor there is the possibility that several faults belonging to different VLIW partitions are not distinguishable; here, we present a list of examples of equivalent faults belonging to two or more partitions that are not distinguishable using a software-based method, only. This analysis has been performed using the equivalence classes generated with the flow explained in Sect. 4 (Fig. 2), and it is a sort of motivation for the method explained in Sect. 6.2, where the partitioning of VLIW processor resources is modified in order to reach the maximum diagnosability.

The first case is shown in Fig. 6 and represents a chain of flip-flops spanning the various pipeline stages of the VLIW processor. Let us consider, for example, a flip-flop that contains one of the bits devoted to identify the destination register of a generic instruction: clearly, this value is provided by the decode stage, and it is available in all the following pipeline stages. Each stage thus contains a flip-flop devoted to save this value. Consequently, we have a set of flip-flops connected in a chain. The behaviour of the processor when a stuck-at fault affects one of the inputs or outputs of these flip-flops is always the same, thus making impossible to identify the root cause fault. Hence, all these faults belong to the same equivalence class. Based on the partitioning strategy adopted so far (which mainly assigns to each partition a single stage) these faults belong to different VLIW partitions (i.e., the decode stage, the execute stage, and the write-back stage). Clearly, this decreases the maximum diagnosability attainable by any SBST program.

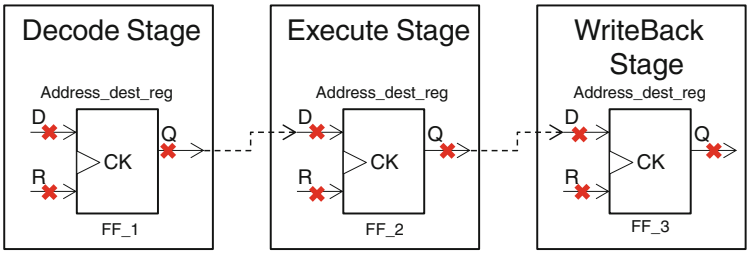

Fig. 6. Example of equivalent faults in different stages of the pipeline; stuck-at faults are highlighted with red x (Color figure online)

The second case is shown in Fig. 7, and it corresponds to the logic gates devoted to the decoding of a generic instruction, and to the flip-flop that contains one bit of the result of the decode operation. Also in this case, if a stuck-at fault affects one of the logic gates embedded in the decoder module or in the flip-flop that stores the result of the decoding, the resulting processor behavior is the same. Hence, it will never be possible to identify if the faulty module is the decoder belonging to the second computational domain or the flip-flop belonging to the generic decode stage module. Consequently, these faults belong to the same equivalence class but to different partitions, and contribute to decreasing the maximum achievable diagnosability.

The third case is shown in Fig. 8, and it corresponds to the register file and the generic flip-flop that contains one bit of the data retrieved from the register file itself. More in particular, both the stuck-at fault affecting the flip-flop of a register and that affecting the flip-flop containing the value of that register in another module (e.g., the generic decode

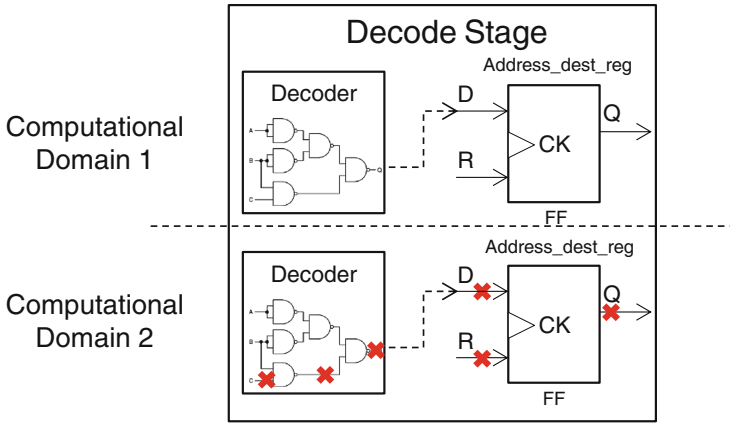

Fig. 7. Example of equivalent faults, considering the decoder module of the second computational domain and the decode stage containing all the instruction decode modules; stuck-at faults are highlighted with red x (Color figure online)

stage module) cause the processor to behave in the same way. Hence, by only checking the signature provided by the diagnostic program, it will never be possible to detect the faulty module; consequently, these faults belong to the same equivalence class. By only checking the signature generated by the diagnostic program explained in Sect. 4 (or by any other test program), it will never be possible to understand if the faulty module is the register file or the decode stage.

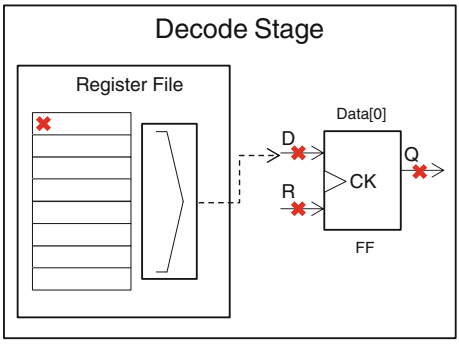

Fig. 8. Example of equivalent faults, considering the register file and the logic resources of the decode stage; stuck-at faults are highlighted with red x (Color figure online)

6.2 Maximization of the Diagnosability of the VLIW Partitions

As described in the previous sub-section, the hardware structure of the VLIW processor is organized in a way that several faults are equivalent wrt the adopted test program and not physically distinguishable using a software-based diagnosis method. In case VLIW partitions are selected on the basis of the VLIW hierarchical structure, equivalent faults

may negatively affect the diagnosability, since they can be located in different partitions not uniquely identifiable in case only one of these faults is excited. Therefore, VLIW partitioning based only on hierarchical module became an ineffective solution if applied to a reconfigurable system, since it is slightly effective to identify a particular portion of the processor to be repaired through reconfiguration. As illustrated in Fig. 9, two partitions related to the VLIW hierarchical modules contain two faults belonging to the same equivalent class making impossible their diagnosability.

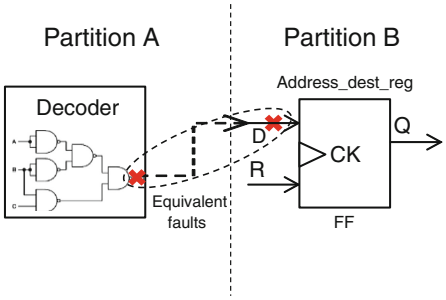

Fig. 9. Example of faults belonging to the same equivalent class traversing two different VLIW partitions

In order to improve the diagnosability, we defined a metric called *Equivalent Cross metric (EC metric)*, which for a given set of VLIW partitions, counts the number of equivalent fault classes crossing two or more partitions (i.e., including faults belonging to two or more partitions). In case the EC metric is nullified, the set of VLIW partitions allows a complete diagnosability of the grouped faults. This result is viable, since the EC metric supports two possible actions. The former consists in the identification of fault groups belonging to a equivalence class traversing more partitions; the latter is the possibility to move the identified equivalence class to a unique partition removing not diagnosable conditions. As an example (reported in Fig. 10), let us consider two faults belonging to a single equivalence class originally related to the partition A and partition B respectively, applying the EC metric we obtain that the two faults are in the single partition A. The minimization of the EC metric allows to obtain a DC metric equal to 100 %, however, the application of this metric is not realistic practicable just by moving equivalence fault classes between the various partitions without considering the logical composition of the VLIW modules, therefore we developed a VLIW partitioning algorithm which is able to take in account the VLIW physical implementation characteristics (e.g., the logical dimension of each VLIW partition) which is depicted in the following sub-section.

6.3 Improved VLIW Partitioning

The equivalence classes generated with the flow described in Sect. 4.2 can be used also to improve the VLIW partitioning, in order to obtain partitions composed of a comparable number of logic resources.

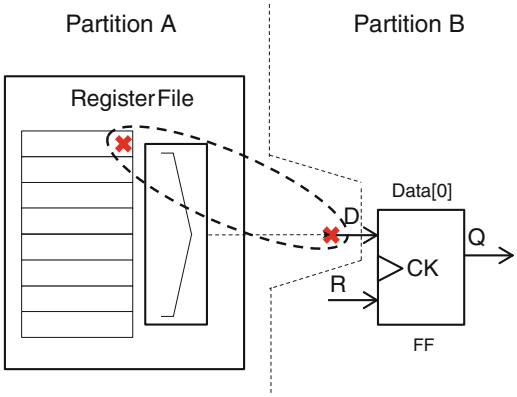

Fig. 10. Example of equivalent fault grouped in a unique partition in order to increase diagnosability

When a device is used in a dynamic reconfiguration environment, and a fault occurs, a diagnostic procedure is required in order to detect the faulty module; moreover, the considered system has to be divided in homogeneous partitions (i.e., having a comparable size), in order to guarantee that the reconfiguration time is always the same.

In Fig. 11 we present an algorithm that, starting from the equivalence classes set, divides a user selected partition P in several partitions, which dimension D is also selected by the user. The first step of the algorithm is aimed at the selection of the set of equivalence classes composed of faults belonging only to the addressed partition P; in this way, each addressed fault has no equivalent faults in any other partition of the considered VLIW processor. In the second step, the set of the new partitions is defined: the user selects how many partitions will contain the resources of the original one. Then, in the step 3, the resources of each equivalence class are iteratively inserted in one of the new partitions. The insertion process starts from the largest equivalence class, in order to guarantee that the equivalence classes composed of a larger number of faults are contained entirely in a single partition. At the end of the algorithm execution, we obtain a set of homogeneous partitions that contain the same resources of the original one.

Considering the diagnosability of the new partitions, the DC metric remains unchanged, since the creation of the new partitions has been done taking into account the equivalence classes: simply, the new partitions have been generated avoiding the allocation of an equivalence classes to more than one partition.

In Figs. 12 and 13 we present the results obtained applying the proposed technique to the ρ-VEX processor, where the register file module has been divided in seven different partitions, denoted as *Register_File_A*, *Register_File_B*, etc. More in particular, Fig. 12 shows the composition of the partitions, in terms of number of faults, before the application of the algorithm proposed in Fig. 11; as it is possible to notice, the number of faults belonging to the register file is high if compared with the one of the others partitions. This peculiarity is not acceptable in a dynamic reconfiguration environment, since if the diagnostic procedure detects a fault in the register file this means that a large

part of the processor has to be reconfigured. Figure 13, instead, shows the results obtained after the module reorganization technique presented in this section; the obtained partitions are more homogeneous than those before the application of the proposed method.

```
1   EC_SET_P = set of equivalence classes composed of faults
    belonging to the addressed partition only;
2   NEW_P_SET = set of new partitions;
3   While EC_SET_P is not empty
    3.1. Select the largest equivalence class EC_MAX from the
         EC_SET_P;
    3.2. Find a new partition NP from the NEW_P_SET with free
         positions larger than EC_MAX.size;
    3.3. EC_MAX is assigned to NP;
    3.4. NP.FreePositions = NP.FreePositions - EC_MAX.size
    3.5. Delete EC_MAX from EC_SET_P;
```

Fig. 11. The pseudo-code for the redistribution of the faults belonging to a single large partition to several partitions

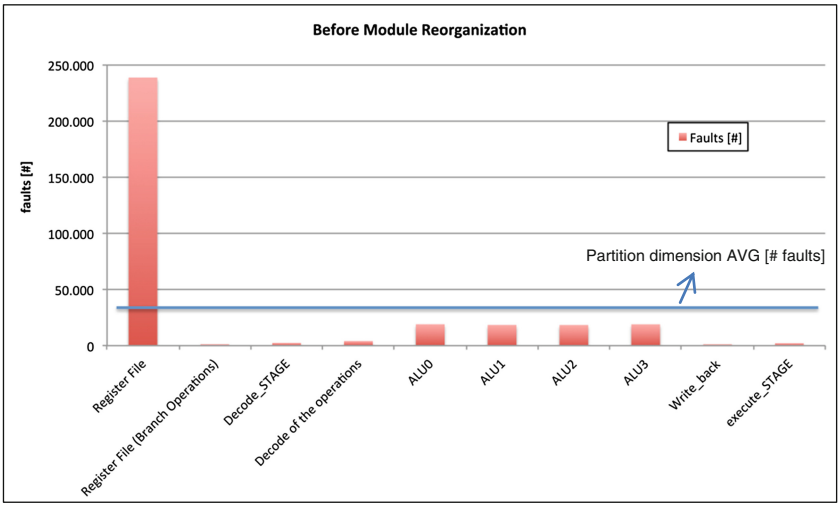

Fig. 12. Fault distribution in the different VLIW partitions before the module reorganization phase based on the equivalence classes

In conclusion, adopting the techniques proposed in this section and in Sect. 6.2, we obtain a complete diagnosability of the considered VLIW partitions; moreover, since the obtained partitions are quite homogeneous in terms of size, the required reconfiguration time of the partitions itself is almost the same.

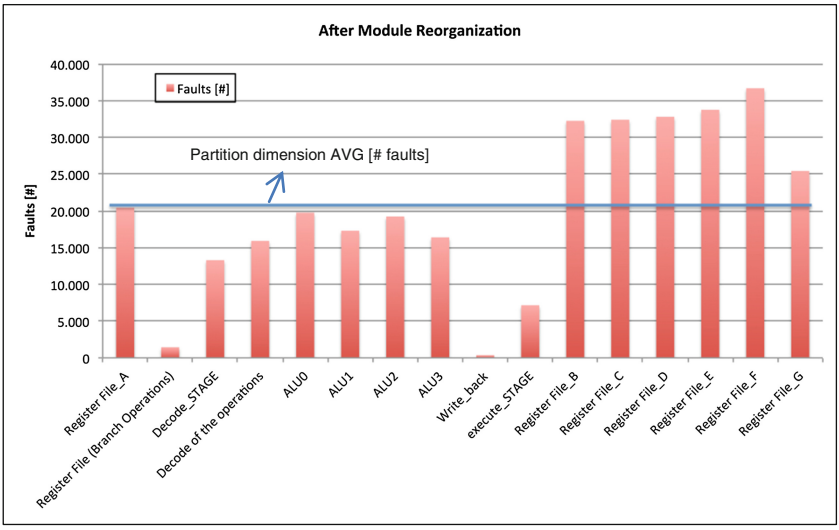

Fig. 13. Fault distribution in the different VLIW partitions after the module reorganization phase based on the equivalence classes

7 Conclusions and Future Work

In this chapter we presented a new method that starting from existing detection-oriented programs generates a diagnosis-oriented test program for a generic VLIW processor. The method exploits the parallelism (and the presence of several alternative resources) intrinsic in VLIW processors to enhance the original test program. The resulting diagnostic program is thus able in most cases to identify the faulty module and is therefore highly suitable for being used within reconfigurable systems. Moreover, we demonstrated that using the equivalence classes generated by the software-based approach, it is possible to maximize the diagnosability of the modules composing the VLIW processor under test.

As future work we plan to estimate the overall performance overhead introduced by the proposed approach and to apply the proposed method to a self-repair system based on reconfigurable device.

References

1. Cardoso, J.M.P., Hübner, M. (eds.): Reconfigurable Computing: From FPGAs to Hardware/Software Codesign. Springer, New York (2011)
2. Fisher, J.A., Faraboschi, P., Young, C.: Embedded Computing: a VLIW Approach to Architecture, Compilers and Tools. Morgan Kaufmann, San Francisco (2004)
3. Bolchini, C.: A software methodology for detecting hardware faults in VLIW data paths. IEEE Trans. Rel. **52**(4), 458–468 (2003)

4. Sabena, D., Sonza Reorda, M., Sterpone, L.: On the optimized generation of software-based self-test programs for VLIW processors. In: IFIP/IEEE 20th International Conference on Very Large Integration System Chip, pp. 129–134 (2012)
5. Psarakis, M., Gizopoulos, D., Sanchez, E., Sonza Reorda, M.: Microprocessor software-based self-testing. IEEE Des. Test Comput. **2**(3), 4–19 (2010)
6. Kabiri, P.S., Navabi, Z.: Effective RT-level software-based self-testing of embedded processor cores. In: IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 209–212 (2012)
7. Sabena, D., Sonza Reorda, M., Sterpone, L.: On the automatic generation of optimized software-based self-test programs for VLIW Processors. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **22**(4), 813–823 (2014)
8. Koal, T., Vierhaus, H.T.: A software-based self-test and hardware reconfiguration solution for VLIW processors. In: IEEE Symposium Design Diagnostic Electronic Circuits Systems, pp. 40–43 (2010)
9. Bernardi, P., Sanchez, E., Schillaci, M., Squillero, G., Sonza Reorda, M.: An effective technique for the automatic generation of diagnosis-oriented programs for processor cores. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **27**(3), 570–574 (2008)
10. Scholzel, M., Koal, T., Vierhaus, H.T.: An adaptive self-test routine for in-field diagnosis of permanent faults in simple RISC cores. In: IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 312–317 (2012)
11. Sabena, D., Sonza Reorda, M., Sterpone, L.: On the development of diagnostic test programs for VLIW processors. In: IFIP/IEEE 21th International Conference on Very Large Integration System Chip, pp. 84–89 (2013)
12. Wong, S., Van As, T., Brown, G.: ρ-VEX: a reconfigurable and extensible softcore VLIW processor. In: International Conference on ICECE Technology, pp. 369–372 (2010)
13. Sabena, D., Sterpone, L., Schölzel, M., Koal, T., Vierhaus, H.T., Wong, S., Glein, R., Rittner, F., Stender, C., Porrmann, M., Hagemeyer, J.: Reconfigurable high performance architectures: how much are they ready for safety-critical applications. In: 19th IEEE European Test Symposium (ETS), pp. 175–182, May 2014
14. Abramson, J., Diniz, P.C.: Resiliency-aware scheduling for reconfigurable VLIW processors. In: International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1–7 (2012)
15. Holst, S., Wunderlich, H.-J.: Adaptive debug and diagnosis without fault dictionaries. In: IEEE European Test Symposium, pp. 7–12 (2007)
16. Ryan, P.G., et al.: Fault dictionary compression and equivalence class computation for sequential circuits. IEEE International Conference on Computer-Aided Design, pp. 508–511 (1993)