On-Line Testing of a Discrete PID Regulator
A Case Study

Uroš Kač, Gregor Papa, Franc Novak, Jurij Šile
Jožef Stefan Institute, Ljubljana, Slovenia

Abstract

Most digital circuits synthesized from a finite-state machine contain idle functional units (modules) at some time during operation in the sense that they do not contain any valid data. In a synchronous circuit with an on-chip synthesized controller which explicitly determines the flow of data, the idle functional units are known in advance for each FSM stage. A case study of the implementation of an on-line built-in self-test that exploits the idle modules is presented in this paper. A concurrent test data-path which cycles test vectors through the idle functional units and produces a response signature is synthesized along with the original algorithm, avoiding test-time overheads.

1. Introduction

The advances in the area of high level synthesis tools and the introduction of CAD systems in the production of ICs brought to a vast increase of the complexity of synthesized circuits. Efficient testing of such circuits is an important issue of the design, production and maintenance process. Both testing and CAD research literature have seen an increased number of contributions involving the topic of design for testability related to high level synthesis. A number of new methods aiming to the enhancement of circuit testability by interventions in the early stages of the design process have been presented, such as hierarchical testing, partial scan paths, and the analysis of high level descriptions to determine and correct hard-to-detect faults through test statement insertion. Other approaches relate to incorporation of scan and partial scan paths into the overall penalty function calculations [1] and to efficient transforms aimed to the inclusion of area-efficient scan paths [2].

In the case of critical systems with high functional reliability (such as production control, phone central offices, etc.), the testing problem represents an even more significant issue. Such systems require frequent or even permanent operation testing and on the other hand they usually do not allow longer system downtimes. This is in contrast with conventional testing methods, like [3], which require part or all of the system to be temporarily taken out of service while a built-in self-test (BIST) or an external test procedure takes place. Despite methods for reducing scan paths [2] have been proposed, conventional approaches are usually unsatisfactory when performed on such systems.

A new approach has been introduced with methods for concurrent or on-line testing of circuits. We could roughly divide these methods into two groups, one concerning methods for on-line verification of the controller operation, and the other including techniques for on-line testing of the data-path. New proposals related to the latter have been introduced by [5] and [6], exploiting the so called functional unit (FU) idle time to perform a data-path test. While [5] describes a universal strategy for the test design process, in [6] an algorithm for the synthesis of
the test data-path, based on a data-flow graph, is presented.

This paper describes a case study of the concurrent data-path test implementation during high level synthesis. The test itself is rendered as a parallel process, with the goal of testing all the FUs and their interconnections with registers, without affecting the number of FUs and the normal operation of the circuit. The use of a pseudo-random pattern generator (PRPG) for test vector generation, and response compression by multiple input signature register (MISR) to avoid storing a large number of comparison vectors are proposed. Possible advantages of concurrent testing are a faster response to a system fault and the efficiency of at-speed testing of the circuit.

The high level synthesis system on which the project is based and the target architecture are presented briefly in section 2. The case of a PID process regulator and the appropriate algorithm are described as well. The testing structure design process is described in section 3. The results of the synthesis process and the comparison of the circuit featuring the on-line BIST with the original circuit are presented in section 4. The last section denotes some problematic aspects that arose during the on-line test incorporation process.

2. The high-level synthesis system and the target application

AMICAL [7], a product of INPG/TIMA, Grenoble, France, is a high level synthesis system targeted towards control-flow dominated machines. Starting with a pure VHDL input at the behavioral level, it produces a full specification for existing logic and register transfer level (RTL) synthesis tools. AMICAL uses a powerful scheduler that transforms a VHDL behavioral description into a FSM presented as a transition table. Allocation of FUs, described in an external library, and synthesis of the output specification in a form of controller-datapath structure are performed afterwards. The system allows to evaluate the design by a built-in penalty function considering different constrains (such as area, power dissipation, number of used FUs, multiplexers/buses). The designer can choose between two possible topologies as a target structure: random topology (MUX-based) or linear topology (BUS-based). We have chosen the latter in the case of the PID regulator.

![Figure 1](image)

Figure 1

We implemented the on-line BIST on a simple algorithm of a discrete PID regulator. Figure 1 illustrates the block diagram of a regulated process. The quadratic rule integration method has been applied for solving the system equation obtaining the following formula:

\[ u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \]  \hspace{1cm} \text{Eq. a}

where \( q_0 = K(1 + T_p / T_0) \), \( q_1 = -K(1 - T_p / T_1 + 2 T_d / T_0) \), \( q_2 = K T_d / T_0 \). Due a further generalisation of the algorithm the PID input values were simplified to \( K_p = K \), \( K_i = K / T_i \), \( K_d = K T_d \) and the adequate VHDL behavioral description, Figure 2(a), was composed.
We shall emphasize at this point that the way of implementation of the behavioral description itself can be of great importance for the further stages of the design process. The impact of the initial form of the circuit description on the fulfillment of constrains posted by area or speed demands however won’t be amongst the topics of this paper.

3. Test data-flow graph design

The FSM description obtained by the scheduler can be presented in a data-flow graph (DFG) form, Figure 2(b). It is then easy to identify, on a state-time basis, all the FUs and registers that do not contain any valid data during specific stages and are therefore available for a possible concurrent test, Figure 2(c). The assumption has of course been made that the circuit is completely synchronous. A possibility exists that the idle FUs can be connected into a parallel test data-path, which would duplicate all the original interconnections between FUs and registers. Therefore the original register allocation would have to be changed and new registers added to the design while the number and the original allocation of FUs should remain intact. The number of interconnections (multiplexers or buses) shall increase as well. A PRPG and a MISR have to be included in the circuit at convenient points. As it is obvious from the VHDL description the circuit will run in an infinite loop during normal operation mode. Therefore also the test data-flow graph (TDFG) can be created such that a loop is formed. In this case a new response value to the test vectors will be added to the MISR at each transition through the loop. Since the test will be concluded only after the completion of the PRPG sequence, the actual TDFG loop will be much longer than the original DFG loop. The PRPG can be also used as a

218
loop counter which can initiate the adequate hardware procedures for signature comparison and test reset after the completion of its sequence.

During the initial step of test data-flow graph construction we have to assign a separate register to each variable in each FSM stage, Figure 3(a). On the other side, temporary input and output registers have to be allocated to idle FUs forming the TDFG, Figure 3(b). The registers will later be merged to obtain a final structure including a much smaller set of registers. We have to keep in mind that the output registers cannot merge within the same FSM stage, while the input registers can. For example, register $a$ can merge with register $c$ or $d$, or even with register $b$ if we are willing to accept some loss of randomness in the test vectors.

We can proceed by listing the registers accordingly to their appertainance to FUs's inputs and outputs. Each register - FU input/output pair in the DFG forms a data transfer path that has to be tested, while the analogous pairs in the TDFG denote the possible test sequences. For the multiplier (FU1) in the PID example we can compose:

$$FU_{\text{left input}} = \{R8, R17, R24, R37, R41\} \rightarrow \{m, q, b, l, b\} = FU_{\text{test left input}}$$

Eq. b)

$$FU_{\text{right input}} = \{R10, R21, #2, R40, R44\} \rightarrow \{n, ar, bm, bv\} = FU_{\text{test right input}}$$

$$FU_{\text{output}} = \{R16, R25, R32, R42, R46\} \rightarrow \{j, u, au, br, cc\} = FU_{\text{test output}}$$

We have subscribed appropriate FSM stages in which registers are written to. As the FSM stage $S1$ of the PID example is only active at circuit reset, possible test registers bounded to

219
that state ($a$ and $b$ in Figure 3(b)) should not be used in TDFG. The constant value #2 is hard wired into the circuit, hence it cannot merge with any register.

Instead of duplicating the exact datapath sequences as they appear in the original DFG we could achieve better incidence by testing transfers associated with the FU separately. Thus we can test the transfer from the input register into the FU and the transfer from the FU into the output register individually.

We can establish from Eq. (b) that the exclusion of registers $a$ and $b$ from the list leads to the necessity for at least two registers from the left side to merge, such to achieve full coverage of the DFG by TDFG. A graphical register-merge algorithm [6] helps us construct the TDFG. The register-merge graph is a graphical representation of the relations between the registers in the data-path and those in the test-path. After composing lists similar to Eq. b) for the remaining three FUs we can begin with the register merging process. Besides the necessity of merging registers to duplicate all data transfers of the data-path in the test-path, we would also like to merge registers to reduce circuit area. A heuristics [6] orders the register mergers on the basis of an estimated saving of buses and MUXs, giving the following priorities:

1. Transfers to/from registers that have the same source and the same destination.
2. Transfers to registers with the same source.
3. Transfers from registers with the same destination.
4. Register-to-register transfers.
5. Transfers between the same FUs with the source and destination interchanged.
6. Transfers with nothing in common.

However the primary objective of the register merging is to satisfy the constrains set out by testability considerations. The testability restrictions might impede some mergers or force others. From these, further constrains might be propagated on subsequent mergers. This can lead the choice of stages for a test transfer to be reduced to zero. Such a merger is then recorded as impossible. For the PID example the resulting DFG and TDFG are shown in Figure 4(c) and 4(d) respectively. The PRPG and MISR have been added to the TDFG at suitable points.

4. The output structure

After the new DFG and the appropriate TDFG had been obtained, an inverse transformation to the FSM in the transition table form had to be carried out. The latter was then fed into the AMICAL synthesis system along with an external library of functional units. The allocation of FUs and busses resulted in a new controller-datapath structure. A graphical representation of the synthesised datapath is presented in Figure 5(a) while the original circuit datapath is illustrated in Figure 5(b). A comparison between the two, Figure 5(c), indicates an increase in the number of registers and buses in the tested circuit whereas the number of FUs is equal in both the original and tested circuit. The increase in the controller area should be considered as well since the number of data transfers during single FSM stages in the tested circuit increased drastically. The realization of the PRPG and the MISR also has certain impact on the circuit area.

5. Conclusion

We have to notify the reader that data transfers are not 100% tested, since we intentionally left out of the TDFG some register-to-register transfers. It proved later in the merging process that merging of registers involved in register-to-register transfer is frequently in contrast with merging of registers involved in register-to-FU (and vice versa) transfers. Tentatives to include all register-to-register transfers into TDFG resulted in further increasing of number of registers.
or even proved impossible without affecting the length of the original DFG. On the other hand, new register-to-register transfers have been introduced in the TDFG to allow the formation of the test data-path. From the test point of view this should only cause some overhead tests to be performed, while from the circuit area point of view this means an increase in the number of interconnections and in the controller area.

Further work is required on the aspect of fault coverage of the synthesized circuit. Improvement of the merging algorithm as well as the issue of compromise determination between the testing and other circuit design constrains is amongst our future goals too.

Acknowledgements

This work was supported by the Slovenian Ministry of Research and Technology under grant 12-7258.

The authors also wish to thank the Laboratory TIMA for providing us the AMICAL package and for their technical support.

References


