

High Level Modifications of VHDL Descriptions for On-Line Test or Fault Tolerance

R. Leveugle, R. Cercueil
TIMA Laboratory
46, Avenue Félix Viallet - 38031 Grenoble Cedex - FRANCE
E-mail: Regis.Leveugle@imag.fr

Abstract

With the increasing probability of transient faults such as bit-flips due to SEUs and the increasing complexity of integrated circuits, the need for integrated mechanisms providing on-line error detection or fault tolerance is becoming a major concern, not only for classically critical applications, but also for circuits used in everyday life. This paper reports on a tool automating the implementation of some mechanisms by inserting modifications in high-level VHDL descriptions. The modifications are compatible with industrial design flows based on commercial synthesis and simulation tools. Implementation results are presented and compared with results previously obtained using a specific synthesis tool.

1. Introduction

It is today well established that the evolution of the CMOS technologies, combining decreasing propagation times, lower power supply and decreasing node capacitances, makes the integrated circuits increasingly sensible to their environment. Phenomena reported for years in the space environment, such as bit-flips due to the impact of particles (called single event upsets, or SEUs), are becoming now observable even at the sea level, due to the impact of atmospheric neutrons. With the increasing probability of transient faults such as SEUs and the increasing complexity of the circuits, the need for integrated mechanisms providing on-line error detection or fault tolerance is becoming a major concern, not only for classically critical applications, but also for circuits used in everyday life. Such mechanisms have been proposed for years, involving very different approaches ranging from transistor-level structural modifications to dedicated high-level synthesis algorithms. However, such mechanisms are not yet widely implemented in industrial circuits.

One of the main reasons explaining the low level of application of the proposed approaches is the lack of tools and/or the lack of compatibility of these tools with commercial tools and industrial design flows. Some tools were actually proposed (e. g. [1, 2]) but remained internally used. The specific state assignment option provided in the Autologic synthesis tool [3], although in a commercial tool, was not flexible enough to be widely used. In consequence, most of the circuits implemented with on-line detection or tolerance mechanisms require noticeable development time, that is not compatible with the time-to-market pressure in most application areas. In such circuits, the specific mechanisms are mostly implemented manually or with proprietary tools [4].

The goal of the study reported here is to cope with these limitations by developing a tool allowing the designer to easily insert, in a digital circuit, specific devices for on-line detection or tolerance of errors provoked by SEUs. The automated approaches must be flexible enough to provide answers for various application constraints (in terms of area, speed, power consumption and error coverage). Several approaches are therefore currently considered. In particular, the architectures presented in [2] proved to provide interesting trade-offs. These architectures, previously implemented at the gate level during the logic synthesis process, are

revisited to evaluate the feasibility of their implementation earlier in the design flow. The tool provided to automate this implementation must be compatible with industrial design flows based on commercial synthesis and simulation tools ; we therefore focus on a stand-alone tool able to modify synthesizable VHDL descriptions. The modified description must of course be optimised for an efficient synthesis.

The state registers are known to be one of the most critical targets for SEUs, since a bit-flip in such a register can provoke unpredictable effects. We therefore currently focus on architectures allowing to protect the circuit by detecting or tolerating SEUs in state registers or in the logic computing the next state, so that erroneous transitions can be tolerated or at least a safe state can be activated. In the current implementation of the tool, the modifications are performed on circuit blocks described either as a Finite State Machine (FSM), or as a Register-Transfer Level (RTL) control flowchart. We also focus on the implementation of dedicated dependable architectures, providing a better trade-off between overheads and coverage than the classical massively redundant architectures (using duplication or triplication and voting). The selected architectures were presented in [2]. One aims at the on-line detection of transition errors and performs control-flow checking using signature analysis. The other aims at tolerating single faults in the state register and in the next-state logic. These two architectures impose specific constraints on the VHDL descriptions and on the synthesis process. We therefore think that they are significant examples, allowing us to conclude on the feasibility of automatic high-level VHDL modifications for dependability increase. Let us notice that this is the first attempt, up to our knowledge, at automating specific fine-grained redundancy insertion by modification of high-level descriptions. Previously published work used massive redundancy or only a classical information redundancy based on error detecting/correcting codes to locally protect registers [5].

The targeted architectures are briefly presented in sections 2 and 3. Section 4 summarizes the approach used for automatic modification of high-level VHDL descriptions. Section 5 finally discusses experimental results.

2. Fault tolerance

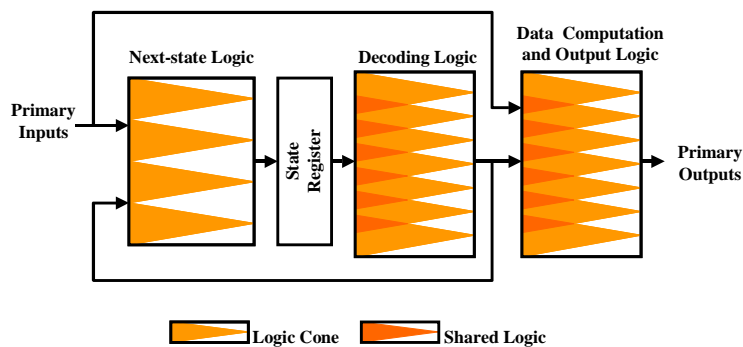


Figure 1: The SID architecture, aiming at tolerating single bit-flips in a state register [2].

Figure 1 illustrates the selected architecture for fault tolerance, named SID (Single Independent Decoder) [2]. This architecture aims at tolerating single bit-flips in a state register. Such a bit-flip can be caused by a SEU on the state register flip-flops, but also by a SEU in the combinatorial logic computing the next state. This part of the logic has therefore to be implemented with independent logic cones, so that a single fault in the logic can only cause a single erroneous bit in the state register. Such an implementation thus requires a precise control of the synthesis process to avoid sharing gates between the cones. The states are encoded using a Hamming single error correcting (SEC) code, as initially proposed by Armstrong [6]. The fault tolerance is achieved by a decoding logic connected onto the outputs of the state register.

This decoder corrects an erroneous bit in the current state code, before this code is used to compute the outputs and the next-state code. In this architecture, the Hamming decoding logic remains a critical block since a single fault in it may lead to an unrecoverable error. The area of this block must therefore be carefully optimised, so that the probability of a fault in it is minimised.

3. On-line detection

Figure 2 illustrates, on the example of a FSM implementation, one of the architectures considered for on-line transition error detection. This architecture is called CFC-AJS because it is based on control-flow checking (CFC) using signature analysis and explicit signature adjustments (AJS) to ensure an easy-to-verify invariant property on the signature. This implementation is not limited to the detection of single bit-flips in the state register ; multiple bit-flips due to a particle impact in the combinatorial logic can also be detected.

This architecture basically works as follows [2]. The designer defines the critical states in the circuit, on a functional basis. These states will be those on which a verification of the sequencing will be performed ("check points"). A monitor is then added to the circuit, including a Multiple-Input Linear Feedback Shift Register (MISR) and some logic, as shown in Figure 2. The MISR computes a signature by compacting the new current state code at each clock cycle. This signature is representative of the path followed through the control flow graph, i.e. of the sequence of states reached during operation. The approach aims at detecting illegal paths, defined as state sequences that are not possible in the circuit specification. In order to achieve an easy identification of an illegal sequencing, an invariant property is forced onto the signatures: the compacted state code is "adjusted" through a XOR function so that the signature obtained in the MISR is always the same on a given state, irrespective of the path followed through the graph to reach this state. The adjustment is equal to zero (i.e., no modification of the code is made) when the state has only one predecessor. A specific adjustment value is computed on N-1 paths for each state having N predecessors. The hardware overhead due to the adjustment computation logic is minimised by optimising the choice of the transitions on which an adjustment is actually made. Furthermore, some gates can be shared with the next-state computation logic and the output computation logic.

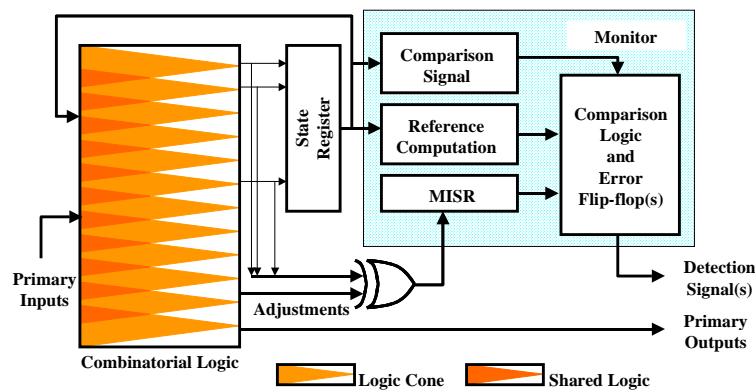


Figure 2: The CFC-AJS architecture, based on signature analysis and aiming at on-line detection of erroneous transitions [2].

Checking then consists, when the current state corresponds to one of the selected check points, in comparing the runtime signature with the invariant reference associated with the current state code. The references are pre-computed and can generally be generated at runtime using a very simple combinatorial logic implemented in the monitor (in the particular case of a

single checkpoint, the reference is just a constant on the input of the comparator). The comparison logic generates an error signal which can be, for example, encoded using a dual-rail code. The error flag is memorised when the comparison signal indicates that a check point is reached.

Another similar architecture, called CFC-NoAJS [2], can be alternatively used. This second architecture is based on the same principles but does not use explicit adjustment values to ensure the invariant property on the signatures. Instead, implicit adjustments are made by means of a clever specific state assignment. Except the state assignment itself, the architecture CFC-NoAJS is simpler because no logic is required to compute the adjustment values and the next-state code can be directly connected to the input of the MISR. In consequence, less modifications have to be made to the initial VHDL description. We will therefore focus in the following on the CFC-AJS architecture, which requires more complex modifications of the high-level description.

4. A CAD tool for automatic modification of high-level descriptions

There is a strong relationship between the modifications in the VHDL descriptions and the synthesis process, since it must be ensured that the synthesis tool will not partially or totally suppress the redundancy inserted in the description. Also, some information may be necessary from the synthesis (e.g. the state assignment result) and some control may be necessary onto the synthesis process (e.g. to generate independent logic cones in the SID architecture). A global implementation flow, including the interactions with the synthesis tool, was therefore defined for the architectures presented in sections 2 and 3, before developing a tool performing automatically the modifications in the high-level descriptions of the circuits. The flow and the VHDL modifications have been defined so that they are independent from the synthesis tool ; any commercial tool can thus be selected by the designer.

The generation of the modified circuit description for the CFC-AJS architecture is performed in several phases:

- In the case of the CFC-AJS architecture, the first and most complex task is the analysis of the control flow graph to determine the locations of the adjustments. The algorithm used is the one previously implemented in the ASYL-SdF tool [2] and it will not be detailed here since it is not directly related to the VHDL modifications.
- A state assignment is then performed on the initial description using the synthesis tool. Of course, this phase is necessary only if the states are specified as an enumerated type, without any user encoding. Any user-defined state assignment can also be used.
- Once the state assignment is known, the signatures associated with each state and the adjustment values are computed with respect to a MISR specified by the user. Currently, we use a MISR with internal XOR gates (the so-called modular structure) and a divisor polynomial either specified by the designer or automatically selected in a list of primitive polynomials. However, this could be easily changed.
- The initial circuit description is then modified to insert the adjustment computations. The definition of the enumerated type used for the states is also extracted and put into an external package, so that it can be used in the description of the monitor. A RTL description of the monitor is then generated according to the user-defined check points. Finally, the complete hierarchical description of the circuit with on-line test is obtained by connecting the modified circuit description and the monitor description.

The VHDL modification for the SID architecture requires more precautions to avoid unwanted logic minimisations during synthesis. The different phases are summarized hereafter.

- The initial description is first loaded into the synthesis tool and a state assignment on a minimal number of bits is performed. Of course, this phase is necessary only if the designer does not pre-define any encoding.

- Once the state assignment is known, the state codes are completed by adding the Hamming check bits.
- As mentioned in section 2, the logic computing the next state must be implemented with independent logic cones. This is achieved by making a hierarchical decomposition of the circuit, with separated blocks for each state register input. This approach avoids a complex control of the synthesis process, which may furthermore have to be adapted to each synthesis tool, depending on the available options.

The initial description is therefore analysed to separate the next state computations from the other statements, related to data or output computations. The next state computations are further decomposed into computations of next state code bits, taking into account the Hamming state encoding obtained in the previous phase. The modified VHDL description is then generated hierarchically with a sub-block (or VHDL component) for data and output computations, a sub-block corresponding to the Hamming decoding logic, a sub-block for the state register and a sub-block for the computation of each next-state bit. In that way, the description can be correctly synthesized with any tool, without requiring any particular option but respecting the hierarchy during the synthesis, i.e. synthesizing and optimising separately each block. The different blocks may also be synthesized with different options, depending on the designer goals. For example, the critical decoding logic can be optimised for area while some other blocks can be optimised for speed.

Let us emphasize that for the two architectures many different modifications have to be made in a classical RTL description. Changing the state assignment in the SID architecture to get error-correcting capabilities would be quite easily done by a designer, using a manual assignment and a "dont_touch" option on this assignment during the synthesis. However, no commercial synthesis tool would add the decoding logic required to achieve fault tolerance. Adding manually this logic (in the initial description or after synthesis) would noticeably impact the circuit development time. Synthesizing independent logic cones in the next state logic and minimised logic in the other parts is also impossible with commercial synthesis tools if the RTL description does not take explicitly this constraint into account. The proposed tool can therefore noticeably help in reducing the design time.

5. Experimental results

It was proved that the architectures CFC-AJS and SID can lead to lower overheads (in terms of area and power consumption) than more classical architectures and also to very interesting dependability characteristics (in terms of fault coverage, mission time or mean time to first failure). These results were obtained when inserting the extra logic during the synthesis and carefully optimising each added element. The drawback of this approach was to require a specific synthesis tool, namely the Asyl-SdF tool [2]. The experiments reported here aimed therefore at comparing the overheads previously obtained with Asyl-SdF and those obtained with a commercial synthesis tool after the proposed modifications in the high-level circuit specification.

The first evaluations performed on simple circuit examples showed that similar area overheads were attainable with the two approaches. This encouraging result led us to implement the tool presented in section 4. Using this tool, the approach is being extensively evaluated using the classical benchmark suits and various other circuit examples coming from university and industry. The number of inputs, outputs, states and transitions and the type of some of these benchmarks are given in Table 1. The results for these benchmarks are shown in Table 2 and 3, for the area and critical path overheads of the SID architecture. These results have been obtained with the synthesis tool Leonardo from Mentor Graphics and the library cub in the CMOS 0.6 micron technology from AMS. All the results for the proposed approach have been obtained using the "binary" state assignment option in Leonardo, that generates state

codes on a minimal number of bits. The results used as reference had been obtained with the specific synthesis tool Asyl-SdF and the CMOS 1.2 micron standard cell library from VLSI Technology [2]. For the approach proposed in this paper, the figures have been obtained after synthesis, using estimated interconnection characteristics. The reference results were obtained after placement and routing in the Compass environment. These comparisons may therefore be refined, taking into account for the two approaches the actual interconnection characteristics after placement and routing. However, the experiments made on some circuit examples showed that the information available after synthesis are either pessimistic or quite close to the results obtained after placement and routing. As examples, the area overhead after placement and routing for the benchmarks called "jay" and "dstate" are 50% and 68% respectively, instead of 79.01% and 103.33% estimated after synthesis. The main conclusions of the study would therefore be similar. Furthermore, the figures obtained cannot be strictly compared, since the technologies, tools and libraries are different. We can only compare the order of magnitude of the overheads.

The results obtained clearly show that the developed methodology is, on an average, as efficient as the approach in [2] for the SID architecture. 66 benchmarks have been currently modified and synthesized. Only 30 out of the 61 benchmarks previously synthesized using Asyl-SdF had higher area overheads using high-level modifications. For the 66 examples, the average area overhead is equal to 136% with the proposed approach and a synthesis optimising the circuit area, to be compared with a 149 % average overhead in the previous approach. When performing a synthesis optimising the critical path of the circuit, the average area overhead is 150% with the proposed approach.

In terms of critical path, the average overhead estimated by the synthesis tool is 55% (61% when the synthesis aims at optimising the delay), to be compared with a 198 % overhead (after place and route) in the previous approach.

Table 4 and Table 5 show results obtained for the CFC-AJS architecture using the same tools and libraries. As in the previous case, these results show the efficiency of the proposed approach. The average area overhead on the 66 benchmarks is 115.38% when the synthesis optimises the area and 108% when optimising the critical path (but the overhead is for example as low as 7.82% for the benchmark imec7). This can be compared with an average overhead equal to 130.82% (after placement and routing) in the previous approach. The average critical path overhead is respectively 48.37% and 55.36% (only 0.1% for imec7) for the proposed approach.

Table 1: Benchmark characteristics.

| Benchmark | Inputs | Outputs | Transitions | States | Type |
|-----------|--------|---------|-------------|--------|-------|
| bbara | 4 | 2 | 60 | 10 | Mealy |
| jay | 4 | 33 | 123 | 58 | Moore |
| protocole | 9 | 9 | 26 | 20 | Moore |
| imec1 | 14 | 67 | 226 | 101 | Moore |
| s1 | 8 | 6 | 107 | 20 | Mealy |
| s1488 | 8 | 19 | 251 | 48 | Mealy |
| s27 | 4 | 1 | 34 | 6 | Mealy |
| dstate | 20 | 12 | 63 | 12 | Mealy |
| dk27 | 1 | 2 | 14 | 7 | Mealy |
| ex_moore | 2 | 3 | 16 | 8 | Moore |

Table 2: Area and critical path overheads for the SID architecture, after a synthesis optimizing the area or the critical path (evaluated from results after synthesis).

| Overhead Optimization criterion | Area | | Critical path | |
|---------------------------------------|---------|---------|---------------|---------|
| | Area | Delay | Area | Delay |
| bbara | 141.18% | 160.47% | 52.37% | 66.97% |
| jay | 79.01% | 74.07% | 24.17% | 13.34% |
| protocole | 115.52% | 118.67% | 40.64% | 65.77% |
| imec1 | 67.11% | 59.03% | 50.00% | 40.03% |
| s1 | 115.44% | 103.66% | 71.04% | 42.88% |
| s1488 | 45.61% | 49.84% | 53.81% | 20.91% |
| s27 | 238.46% | 293.33% | 152.17% | 124.55% |
| dstate | 103.33% | 90.98% | 54.64% | 39.19% |
| dk27 | 184.62% | 246.67% | 98.83% | 125.93% |
| ex_moore | 221.43% | 227.78% | 81.85% | 46.13% |

Table 3: Comparison of area and critical path overheads for the SID architecture, after a synthesis optimizing the area (evaluated from results after synthesis for the current approach, and from actual figures after placement and routing for the previous approach [2]).

| Overhead | Area | | Critical path | |
|-----------|--------------|---------------|---------------|---------------|
| | Current work | Previous work | Current work | Previous work |
| bbara | 141.18% | 191.94% | 52.37% | 198.46% |
| jay | 79.01% | 94.98% | 24.17% | 152.67% |
| protocole | 115.52% | 130.76% | 40.64% | 334.48% |
| imec1 | 67.11% | 47.58% | 50.00% | 77.15% |
| s1 | 115.44% | 95.29% | 71.04% | 200.00% |
| s1488 | 45.61% | 69.06% | 53.81% | 131.15% |
| s27 | 238.46% | 272.77% | 152.17% | 291.89% |
| dstate | 103.33% | 145.34% | 54.64% | 172.83% |
| dk27 | 184.62% | 168.00% | 98.83% | 261.90% |
| ex_moore | 221.43% | 130.70% | 81.85% | 202.04% |

Table 4: Area and critical path overheads for the CFC-AJS architecture, after a synthesis optimizing the area or the critical path (evaluated from results after synthesis).

| Overhead Optimization criterion | Area | | Critical path | |
|---------------------------------------|---------|---------|---------------|---------|
| | Area | Delay | Area | Delay |
| bbara | 147.06% | 134.88% | 81.55% | 74.17% |
| jay | 58.56% | 54.32% | 14.00% | 19.15% |
| protocole | 81.03% | 72.00% | 6.69% | 32.66% |
| imec1 | 33.22% | 23.79% | 34.31% | 15.80% |
| s1 | 63.97% | 54.45% | 47.48% | 28.64% |
| s1488 | 51.75% | 52.06% | 11.75% | 15.74% |
| s27 | 261.54% | 253.33% | 213.48% | 204.91% |
| dstate | 56.67% | 49.62% | 37.94% | 44.07% |
| dk27 | 230.77% | 206.67% | 138.67% | 142.59% |
| ex_moore | 171.43% | 144.44% | 83.27% | 38.39% |

Table 5: Comparison of area overheads for the CFC-AJS architecture, after a synthesis optimizing the area (evaluated from results after synthesis for the current approach, and from actual figures after placement and routing for the previous approach [2]).

| Benchmark | Area overhead (Current work) | Area overhead (Previous work) |
|-----------|------------------------------|-------------------------------|
| bbara | 147.06% | 227.19% |
| jay | 58.56% | 131.98% |
| protocole | 81.03% | 138.19% |
| imec1 | 33.22% | 31.33% |
| s1 | 63.97% | 127.85% |
| s1488 | 51.75% | N/A |
| s27 | 261.54% | 383.08% |
| dstate | 56.67% | N/A |
| dk27 | 230.77% | 171.56% |
| ex_moore | 171.43% | 150.86% |

6. Conclusion

The feasibility of dependability increase by automatic modifications of high-level VHDL descriptions has been demonstrated and a tool has been developed. The tool can be used to modify circuit blocks described either as finite state machines or RTL control flowcharts. Results have been obtained on a large set of benchmarks coming from various sources and demonstrate the interest of the proposed approach to implement circuits with the SID and CFC-AJS architectures.

This is the first attempt at automating the implementation of such specific dependable architectures by modification of high-level descriptions. Results are encouraging since they are, on an average, very close to those obtained when the modifications are performed by a specific synthesis tool as reported in [2], although in that later case the insertion of the additional elements can be controlled more precisely and therefore can be more optimised. The slight loss in optimisation is compensated by the advantageous compatibility with industrial design flows based on any commercial synthesis and simulation tools. Furthermore, modifying the high-level description allows the designer to evaluate earlier in the design process the global dependability of his circuit. Last but not least, lower overheads have been achieved for a noticeable number of benchmarks, compared with the previous approach.

References

- [1] I. Alzaher-Noufal, M. Nicolaidis, "A CAD framework for generating self-checking multipliers based on residue codes", Design, Automation and Test in Europe Conference (DATE), March 9-12, 1999, pp. 122-129
- [2] R. Rochet, R. Leveugle, G. Saucier, "ASYL-SdF: a synthesis tool for dependability in controllers", IEICE Transactions on Information and Systems, Special Issue on "Synthesis and Verification of Hardware Design", Vol. E79-D, no. 10, October 1996, pp. 1382-1388
- [3] Autologic User Manual, Mentor Graphics
- [4] E. Böhl, R. Stephan, W. Glauert, "The architecture of the fail-stop controller AE11", 3rd IEEE International On-Line Testing workshop, Aghia Pelaghia headland, Crete, Greece, July 7-9, 1997, pp. 47-52
- [5] F. Vargas, A. Amory, "Recent improvements on the specification of transient-fault tolerant VHDL descriptions: a case-study for area overhead analysis", 13th Symposium on Integrated Circuits and Systems Design (SBCCI 2000), Manaus, Amazonas, Brazil, September 18-24, 2000, pp. 249-254
- [6] D. B. Armstrong, "A general method of applying error correction to synchronous digital systems", The Bell System Technical Journal, vol. 40, no. 2, March 1961, pp. 577-593