Time-redundancy Transformations for Adaptive Fault-Tolerant Circuits

Dmitry Burlyaev
Univ. Grenoble Alpes / INRIA
Grenoble, France
dmitry.burlyaev@inria.fr

Pascal Fradet
INRIA / Univ. Grenoble Alpes
Grenoble, France
pascal.fradet@inria.fr

Alain Girault
INRIA / Univ. Grenoble Alpes
Grenoble, France
alain.girault@inria.fr

***Abstract*—We present a novel logic-level circuit transformation technique for the automatic insertion of fault-tolerance properties. The transformations, based on time-redundancy, allow dynamically changes of the level of redundancy without interrupting the computation. The proposed concept of dynamic time redundancy permits adaptive circuits whose fault-tolerance properties can be "on-the-fly" traded-off for throughput. The approach is technologically independent and does not require any specific hardware support. Experimental results on the ITC'99 benchmark suite indicate that the benefits of our method grow with the combinational size of the circuit. Dynamic double and triple time redundant transformations generate circuits 1.7 to 2.9 times smaller than full Triple-Modular Redundancy (TMR). This transformation is a good alternative to TMR for logic-intensive safety-critical circuits where low hardware overhead or only temporary fault-tolerance guarantees are needed.**

I. Introduction

Fault-tolerance is an important research topic in safety-critical systems [1], [2]. While schemes for hardware redundancy techniques have been proposed early [3] and are widely-used in existing synthesis tools [4], [5], time redundancy is much less investigated in circuits and not integrated in EDA tools. However, time redundancy has specific advantages that make its addition within synthesis tools worthwhile. Indeed, if time redundancy trades off throughput for a low hardware overhead, it is also more adaptive than space redundancy (*e.g.,* TMR). It is possible for circuits to operate at full speed and to dynamically switch to a redundancy mode. Different reasons can motivate the change of the redundancy level (and by consequence the fault-tolerance level): (*i*) occasional processing of critical data, (*ii*) sporadic changes in radiation environment (*e.g.,* high solar activity, entering or leaving specific areas like the South Atlantic Anomaly (SAA) [6]).

We propose the automatic insertion of dynamic time-redundant mechanisms at post-synthesis phase. This significantly reduces design efforts and correctness validation. By dynamic, we mean that the resulting circuit can switch between different operating modes, each mode with a different redundancy level (possibly null). We write DyTR^n for the Dynamic Time Redundancy transformation where the maximum redundancy level is n. For instance, the dynamic triple time redundant transformation DyTR^3 produces a circuit with three redundancy modes: a triple redundancy mode that masks any Single-Event Transient (SET) but operates at one third of the nominal throughput, a double redundancy mode that detects any SET but operates at half of the nominal throughput, and a no redundancy mode that operates at the nominal throughput but without any fault-tolerant properties. In conjunction with frequency and power scaling, the proposed approach provides adaptive design options not available before.

The transformations are performed at the gate level, *i.e.,* netlists of AND, OR, NOT gates plus memory cells (flip-flops (FFs)). This level has two main advantages:

- gate-level netlists can be described by an elementary language, which simplifies correctness proofs[1];
- it is easier to prevent synthesis tools from optimizing (undo) our transformations at this late stage, as well as to integrate the circuit transformation in commercial logic synthesis tools.

We consider fault models of the form "at most M SETs within K clock cycles", denoted by $SET(M, K)$. SET subsumes Single-Event Upset (SEU). Even in environments with high levels of ionizing radiations (*e.g.,* space, particle accelerators), this parameter K is considered to be larger than 10^{10} for modern high-frequency electronics [7]. With appropriate redundancy, the proposed technique can mask M errors. However, the most common and realistic fault-model is of the form $SET(1, K)$, which only requires double redundancy (for fault detection) or triple (for fault masking). In this article, we present informally the general approach and focus on double and triple dynamic time redundancy transformations. Each dynamic time-redundancy transformation consists in replacing each memory cell by a *memory block* that supports the dynamic redundancy and implements the voting/detection. A global *control block* is also added to the circuit to provide signals for controlling the voting and the switching between the available redundancy modes.

Section II presents the notations used in the rest of the paper. We describe the principles of our approach in Section III. We describe the transformation for triple dynamic time-redundancy, DyTR^3, in Section IV and sketch DyTR^2 in Section V. Experimental results using the ITC'99 benchmark suite [8] are presented in Section VI. Section VII surveys the related work on time-redundancy for circuit fault-tolerance. Finally, we summarize our contributions and sketch future research directions in Section VIII.

[1] In this paper, we provide only informal proofs but we have already used that language to prove related transformations using the Coq proof-assistant.

978-1-4673-7501-6/15/$31.00

II. Context and Notations

Any digital circuit can be represented in the most general way as in Figure 1. The circuit, which consists of a combinational part and a sequential part, takes a primary input bit vector $\vec{PI}$ and returns a primary output bit vector $\vec{PO}$ at each clock cycle. The combinational part implements some memoryless boolean function φ.

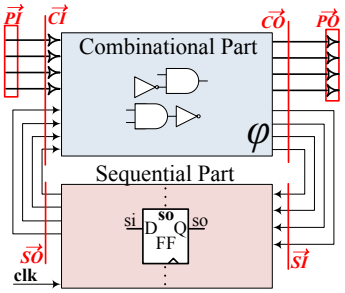

Fig. 1. Digital circuit before the transformation.

We denote the input (resp. output) bit vector of the combinational part by $\vec{CI}$ (resp. $\vec{CO}$) and the input (resp. output) bit vector of the sequential part by $\vec{SI}$ (resp. $\vec{SO}$). They satisfy the following equalities:

$$\vec{CO} = \varphi(\vec{CI}) \qquad \vec{CI} = \vec{PI} \oplus \vec{SO} \qquad \vec{CO} = \vec{PO} \oplus \vec{SI} \qquad (1)$$

where $\oplus$ denotes vector concatenation. We will use lower case (*e.g.,* $\vec{pi}$, $\vec{co}$, etc.) to denote the corresponding signals in the transformed circuits; they satisfy the same equalities. Throughout the paper, we write $\vec{v}_i$ for the value of the bit vector $\vec{v}$ at the i^{th} clock cycle (the numbering starts at $i=1$). Values and outputs of memory cells are denoted by the same names. For instance, the memory cell in Figure 1 with output so is itself denoted so.

An SET can occur on any wire of the circuit and is non-deterministically latched. An SET in a combinational circuit can lead to the arbitrary corruption of any memory cell connected by a purely combinational path to the place where it occurred. A corrupted vector, written $\dagger\vec{v}$, represents the vector $\vec{v}$ with an arbitrary number of corrupted bits. An SET in the combinational circuit of Figure 1 at some cycle i can lead to the corruption of several outputs $\dagger\vec{CO}_i$. This, in turn, can corrupt part of the primary outputs ($\dagger\vec{PO}_i$) and of several circuit's memory cells. This last corruption will propagate at their outputs (*i.e.,* in the $\dagger\vec{SO}_{i+1}$ vector) during the next clock cycle. Note that SETs subsumes the SEUs fault-model since any SEU of a cell can be modeled by a SET occurring at the cell's input and latched.

III. Concept of Dynamic Time Redundancy

Our dynamic time-redundancy transformations DyTR^n transform any circuit at the netlist level according to the following steps:

1) choice of the fault-tolerance properties and of the corresponding operating modes ;
2) substitution of each original memory cell with a *memory block*;
3) addition of a global *control block*;
4) adjusting the design of *input-output interfaces* .

The original combinational part of the circuit is kept unchanged.

Due to space limitations, we do not present the generic transformation (*i.e.,* DyTR^n for all n). In this section, we just outline the main transformation steps. They will be presented in details for the two most common (and useful) instantiations: dynamic triple redundancy (Section IV) and dynamic double redundancy (Section V).

A. Choice of fault-tolerance level and modes

A designer can choose (*i*) the set of needed redundancy levels or operating modes, and (*ii*) the error-detection/masking properties of each mode. For example, an application may ask for either running at full speed (hence no fault-tolerance) or being able to detect up to two simultaneous faults. Thus, only two operation modes (mode 1 and mode 4) are needed. In general, mode 4 may detect one or two faults and mask one fault. In our example, masking is not needed and the design of the mode (and the transformed circuit) can be tuned and simplified.

To match the required fault-tolerance properties, we allocate control signals which, set by the surrounding circuit, will define the current operating mode and support switches.

B. Memory Block

Memory blocks implement the core of the dynamic time-redundant mechanism. They record recomputed results and organize the voting and comparison procedures.

Each memory cell of the original circuit is replaced by a memory block with the same data input and output. The input (resp. output) signals of the sequential part $\vec{si}$ (resp. $\vec{so}$) correspond to the inputs (resp. outputs) of all memory blocks. Memory blocks produce a fail signal whenever an error is detected. Memory blocks also require additional control signals to organize voting and dynamic switches. These signals are produced by a small global Finite State Machine (FSM): the control block.

C. Control Block

The control block is a centralized FSM that provides the control signals used by memory blocks. It can be seen as a collection of circular automata, one for each mode (see Figure 2). The automaton for mode n ($n \in [1 \dots N]$) sets

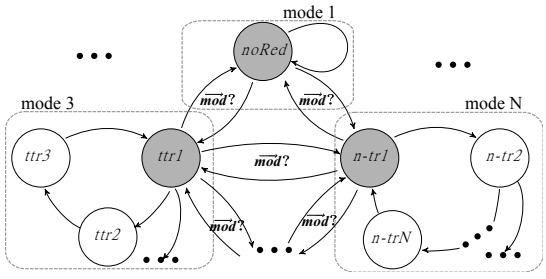

Fig. 2. Control block for the generic DyTR^n transformed circuit

the control signals to select the relevant bits at each step of the n steps of voting/comparison. The control block allocates

special inputs mod used by the surrounding circuit to switch between operating modes. They are used to pass the control from one automaton sub-part to another at specific cycles. In Figure 2, the corresponding states for these cycles are grayed. The control block is itself protected using hardware redundancy, which does not entail a big hardware overhead since it is a small circuit.

D. Input-Output Interface

The environment (or surrounding circuit) should also dynamically upsample and downsample the input/output bits streams of the transformed circuit to support the changes of redundant modes. This requirement can be fulfilled through frequency scaling/division. In particular, when the mode n is chosen, the surrounding circuit should adapt its frequency to produce (resp. consume) n upsampled inputs (resp. outputs).

In many cases, redundancy of inputs and outputs can also be handled by a circuit interface. Assuming a circuit operating in mode n, if the inputs are read from memory, the interface would read a new value every n cycles and duplicate it n times.

IV. Dynamic Triple-Time Redundancy

Dynamic triple-time redundancy DyTR^3 is an instance of the general transformation scheme. It demonstrates all features of the transformations, including error-masking, error-detection, and dynamic modes switch. DyTR^3 offers the following operating modes:

1) no time redundancy and no fault tolerance properties (mode 1).
2) double-time redundancy with a single error detection (mode 2).
3) triple-time redundancy with a single error masking (and an error detection output signal) (mode 3).

A. Memory Block

The memory block for DyTR^3 is represented in Figure 3. It consists of the following components:

- three cells d, d', and d'' form the data bits to save redundant information for voting if $modeS = 3$ and comparison if $modeS = 2$. This pipeline is controlled by the global signal *modeS*: if $modeS = 1$, the cell d is by-passed. This allows to dynamically change the pipeline length and to organize a dynamic delay;
- the cell s (saving line) is needed to have enough redundancy to perform majority voting during three cycles;
- the majority voter *VotA* along with the multiplexers $MuxA$ and $MuxC$ performs error masking and/or error detection. The proper subset of bits for voting/comparison is selected from the delay and saving lines using global control signals $modeS$ and $fetchA$.

The majority voter *VotA* takes three inputs and returns so, the output signal of the memory block. The *Vot/Detect* component also outputs the result of the comparison between d and d' as the *fail* signal.

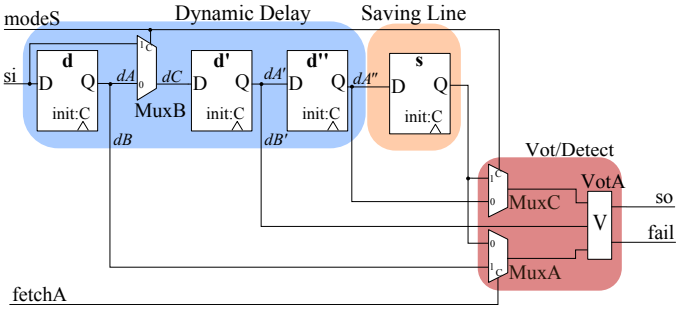

Fig. 3. Memory block for DyTR^3

The internal structure of the *VotA* voter is presented in Figure 4: the *fail* signal is the result of the comparison between a and b. This signal also serves to implement the majority vote by selecting the correct output (c if $a \neq b$; b if $a = b$).

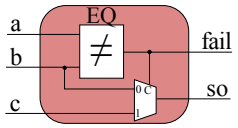

Fig. 4. Voter with detection capability

Hereafter, we describe the functionality of the memory block in each of the three operating modes.

Mode 3

In normal execution (*i.e.,* without SET), the behavior of all memory blocks is described by the following equalities.

In mode 3, the dynamic delay results in:

$$\forall i \in \mathbb{N}^*.\ \ \vec{si}_i = \vec{d}_{i+1} = \vec{d'}_{i+2} = \vec{d''}_{i+3} = \vec{s}_{i+4} = \vec{so}_{i+3} \qquad (2)$$

As described in Section II, the upsampled input/output signals satisfy the same equations as Eq. (1), that is:

$$\forall i \in \mathbb{N}^*,\ \ \begin{cases} \vec{co}_i &= \varphi(\vec{ci}_i) \\ \vec{ci}_i &= \vec{pi}_i \oplus \vec{so}_i \\ \vec{co}_i &= \vec{po}_i \oplus \vec{si}_i \end{cases} \qquad (3)$$

The original input stream $\vec{PI}$ is upsampled three times in this operating mode:

$$\forall i \in \mathbb{N}^*.\ \vec{pi}_{3i-2} = \vec{pi}_{3i-1} = \vec{pi}_{3i} = \vec{PI}_i \qquad (4)$$

From Eqs. (2), (3), and (4), we derive that the output bit stream of the combinational part after the circuit transformation $\vec{co}$ is a three-times upsampled bit stream $\vec{CO}$ of the original circuit.

$$\forall i \in \mathbb{N}^*.\ \vec{co}_{3i-2} = \vec{co}_{3i-1} = \vec{co}_{3i} = \vec{CO}_i \qquad (5)$$

In mode 3, the three cells d, d', and d'' are equal each $(3i-2)th$ cycles as formalized by Eq. (6):

$$\forall i \in \mathbb{N}^*.\ \vec{d}_{3i-2} = \vec{d'}_{3i-2} = \vec{d''}_{3i-2} \qquad (6)$$

Voting on (d, d', d'') at these specific cycles will mask a single error. The result of the vote is forwarded through $\vec{so}$

to the rest of the combinational circuit. The memory cell s is used to save the value d'' for the two subsequent votes. Assuming that d, d', and d'' hold a correct value (say a), the first vote will be between (a, a, a) (stored in (d, d', d'')) and the second between (a, a, a) (stored in (d', d'', s)). In this cycle, d contains the next value of the stream (say b), which will propagate to d'. So, the third vote will be between (b, a, a) (stored in (d', d'', s)). So, if d'' or s is corrupted the vote may return a wrong value, which will be propagated to the next block. Fortunately, this incorrect value is preceded by two correct ones and can be corrected by a special recovery procedure described in details in Section IV-C. As we will see, such an error is corrected within the next six clock cycles.

To support this functionality, the global control signal $fetchA$ is generated by the control block according to Eq.(7).

$$\begin{aligned} fetchA &= 0 \quad \text{at cycles } 3i \text{ and } 3i-1 \\ fetchA &= 1 \quad \text{at cycles } 3i-2. \end{aligned} \tag{7}$$

Mode 3 also implements error detection. In particular, a single bit-flip can be detected each $(3i-2)$ and $(3i-1)$ clock cycles. If no error occurs, the values of d' and d'' memory cells in each memory block must be equal (see Eq. (6)). Otherwise a single bit-flip is detected and the *fail* signal is raised.

It can be shown that any single SET (even on global control signals) will be eventually be masked (see Section IV-C).

Mode 2

The double-time redundant mode is supported by the global control signals $modeS = 0$ and $fetchA = 1$. In normal execution (*i.e.,* without errors), the behavior of the memory block is described by the following equalities:

$$\forall i \in \mathbb{N}^*.\ \ \vec{si}_i = \vec{d}_{i+1} = \vec{d}'_{i+2} = \vec{d}''_{i+3} = \vec{s}_{i+4} = \vec{so}_{i+2} \tag{8}$$

The original input stream $\vec{PI}$ is upsampled twice:

$$\forall i \in \mathbb{N}^*.\ \vec{pi}_{2i-1} = \vec{pi}_{2i} = \vec{PI}_i \tag{9}$$

As a result, the output stream $\vec{co}$ is the output stream $\vec{CO}$ of the original circuit upsampled twice (see Eq. (10)):

$$\forall i \in \mathbb{N}^*.\ \vec{co}_{2i-1} = \vec{co}_{2i} = \vec{CO}_i \tag{10}$$

Error detection properties are based on Eq. (11):

$$\forall i \in \mathbb{N}^*.\ \vec{d'}_{2i-1} = \vec{d''}_{2i-1} \tag{11}$$

In mode 2, the memory cell s does not participate in computation. The vote is performed at each cycle on (d, d', d''). Consider, for example, an upsampled value a_1 and a_2. After two cycles, we have $(d, d', d'') = (a_2, a_1, ?)$ and after three cycles $(d, d', d'') = (?, a_2, a_1)$. Consequently, two votes on (d, d', d'') will produce the expected result twice. Of course, an SET may lead to an error propagation but masking is not guaranteed in mode 2. Note, however, that this mode does perform some (incomplete) masking through voting.

Error detection is organized through the comparison of d' and d'' at odd cycles. If no errors occurs, their values should be equal according to Eq. (11). If the values are not equal, the *fail* signal will be raised to indicate the error detection to the control block. Note that an SET at any wire can corrupt at most one bit in a memory block.

TABLE 1. SWITCHING PROCESS $1 \mapsto 3$.

clk	$\vec{si}$	$\vec{d}$	$\vec{d'}$	$\vec{d''}$	$\vec{s}$	mS	fA	$\vec{so}$
1	$\vec{c}_1$	$\vec{b}_1$	$\vec{b}_1$	$\vec{a}_1$	$\vec{z}_1$	1	1	$\vec{b}_v$
2	$\vec{d}_1$	$\vec{c}_1$	$\vec{c}_1$	$\vec{b}_1$	$\vec{a}_1$	1	1	$\vec{c}_v$
3	$\vec{e}_1$	$\vec{d}_1$	$\vec{d}_1$	$\vec{c}_1$	$\vec{b}_1$	0	1	$\vec{d}_v$
4	$\vec{e}_2$	$\vec{e}_1$	$\vec{d}_1$	$\vec{d}_1$	$\vec{c}_1$	0	0	$\vec{d}_v$
5	$\vec{e}_3$	$\vec{e}_2$	$\vec{e}_1$	$\vec{d}_1$	$\vec{d}_1$	0	0	$\vec{d}_v$
6	$\vec{f}_1$	$\vec{e}_3$	$\vec{e}_2$	$\vec{e}_1$	$\vec{d}_1$	0	1	$\vec{e}_v$
7	$\vec{f}_2$	$\vec{f}_1$	$\vec{e}_3$	$\vec{e}_2$	$\vec{e}_1$	0	0	$\vec{e}_v$
8	$\vec{f}_3$	$\vec{f}_2$	$\vec{f}_1$	$\vec{e}_3$	$\vec{e}_2$	0	0	$\vec{e}_v$
9	$\vec{g}_1$	$\vec{f}_3$	$\vec{f}_2$	$\vec{f}_1$	$\vec{e}_3$	0	1	$\vec{f}_v$

x_v is the result of voting on the values marked in grey at this clock cycle; $mS = modeS$; $fA = fetchA$.

Mode 1

Mode 1 is supported by the global control signals $modeS = 1$ and $fetchA = 1$. The multiplexer ***MuxB*** is used to duplicate the input data in d and d' at each cycle. In normal execution (*i.e.,* without errors), the behavior of all memory blocks is described by the following equalities:

$$\forall i \in \mathbb{N}^*.\ \ \vec{si}_i = \vec{d}_{i+1} = \vec{d}'_{i+1} = \vec{d}''_{i+2} = \vec{s}_{i+3} = \vec{so}_{i+1} \tag{12}$$

The operating with no time redundancy implies that the input streams are not upsampled (see Eq. (13)):

$$n = 1 :\ \forall i \in \mathbb{N}^*.\ \vec{pi}_i = \vec{PI}_i \tag{13}$$

The output of the combinatorial circuit $\vec{co}$ is equivalent to the output of the circuit before the transformation:

$$\forall i \in \mathbb{N}^*.\ \vec{co}_i = \vec{CO}_i \tag{14}$$

From Eq. (12), if no errors occur, d equals to d' each clock cycle. Consequently, voting on three values (d, d', s) returns the value of d (and d') at each cycle. The mode has neither SET masking nor detection properties, but its throughput is comparable to the original circuit before transformation. If d and/or d' is corrupted, then the vote on (d, d', s) may return a wrong value (without raising the *fail* signal).

B. Control block and mode switch

Dynamic triple-time redundancy has three operating modes. In the most general implementation, it can switch from any mode to any other one. The control block governing these switches in presented in Figure 5. We present here only two possible switches.

$1 \mapsto 3$

Switching from the operating mode 1 to triple-time redundancy is performed by setting $modeS$ to 0 and up-sampling the input stream three times. The $fetchA$ signal is raised every three cycles specified by Eq. (7). Table 1 shows such a switch starting from the third clock cycle. The error masking and detection properties are guaranteed only three cycles later (6) when the delay line is filled by three independent redundant bits $(\vec{e}_1, \vec{e}_2, \vec{e}_3)$.

TABLE 2. SWITCHING PROCESS $3 \mapsto 1$.

clk	$\vec{si}$	$\vec{d}$	$\vec{d'}$	$\vec{d''}$	$\vec{s}$	mS	fA	$\vec{so}$
1	$\vec{h}_2$	$\vec{h}_1$	$\vec{g}_3$	$\vec{g}_2$	$\vec{g}_1$	0	0	$\vec{g}_v$
2	$\vec{h}_3$	$\vec{h}_2$	$\vec{h}_1$	$\vec{g}_3$	$\vec{g}_2$	0	0	$\vec{g}_v$
3	$\vec{j}_1$	$\vec{h}_3$	$\vec{h}_2$	$\vec{h}_1$	$\vec{g}_3$	0	1	$\vec{h}_v$
4	$\vec{j}_2$	$\vec{j}_1$	$\vec{h}_3$	$\vec{h}_2$	$\vec{h}_1$	0	0	$\vec{h}_v$
5	$\vec{j}_3$	$\vec{j}_2$	$\vec{j}_1$	$\vec{h}_3$	$\vec{h}_2$	0	0	$\vec{h}_v$
6	$\vec{k}_1$	$\vec{j}_3$	$\vec{j}_2$	$\vec{j}_1$	$\vec{h}_3$	0	1	$\vec{j}_v$
7	$\vec{l}_1$	$\vec{k}_1$	$\vec{k}_1$	$\vec{j}_2$	$\vec{j}_1$	1	1	$\vec{k}_v$
8	$\vec{m}_1$	$\vec{l}_1$	$\vec{l}_1$	$\vec{k}_1$	$\vec{j}_2$	1	1	$\vec{l}_v$
9	$\vec{n}_1$	$\vec{m}_2$	$\vec{m}_1$	$\vec{l}_1$	$\vec{k}_1$	1	1	$\vec{m}_v$

x_v is the result of voting on the values marked in grey at this clock cycle; $mS = modeS$; $fA = fetchA$.

TABLE 3. RECOVERY PROCEDURE - DYTR^3, MODE 3.

clk	$st.$	$\vec{si}$	$\vec{d}$	$\vec{d'}$	$\vec{d''}$	$\vec{s}$	mS	fA	$\vec{so}$
1	$ttr1$	$\vec{b}_1$	$\dagger\vec{a}_3$	$\vec{a}_2$	$\vec{a}_1$	$\vec{z}_1$	0	1	$\vec{a}_v$
②	$ttr2$	$\vec{b}_2$	$\vec{b}_1$	$\dagger\vec{a}_3$	$\vec{a}_2$	$\vec{a}_1$	0	0	$\vec{a}_v$
3	$err1$	$\dagger\vec{b}_3$	$\vec{b}_2$	$\vec{b}_1$	$\dagger\vec{a}_3$	$\vec{a}_2$	0	0	$\dagger a_v$
4	$err2$	$\vec{c}_1$	$\dagger\vec{b}_3$	$\vec{b}_2$	$\vec{b}_1$	$\vec{a}_3$	0	1	$\vec{b}_v$
5	$err3$	$\vec{c}_2$	$\vec{c}_1$	$\dagger\vec{b}_3$	$\vec{b}_2$	$\vec{b}_1$	0	0	$\vec{b}_v$
6	$err4$	$\vec{c}_3$	$\vec{c}_2$	$\vec{c}_1$	$\dagger\vec{b}_3$	$\vec{b}_2$	1	0	$\vec{b}_v$
7	$ttr1$	$\vec{d}_1$	$\vec{c}_3$	$\vec{c}_2$	$\vec{c}_1$	$\dagger\vec{b}_3$	0	1	$\vec{c}_v$
8	$ttr2$	$\vec{d}_2$	$\vec{d}_1$	$\vec{c}_3$	$\vec{c}_2$	$\vec{c}_1$	0	0	$\vec{c}_v$

x_v is the result of voting on the values marked in grey at this clock cycle; $mS = modeS$; $fA = fetchA$.

$3 \mapsto 1$

Switching from three times redundancy to non-redundant mode triples the circuit throughput but eliminates error detection and masking. In Table 2, the switch is performed at the seventh clock cycle by raising signals $modeS$ and $fetchA$.

All other switching scenarios are supported by the DyTR^3 control block as shown in Figure 5. Each circular automaton corresponds to one of the three available operating modes.

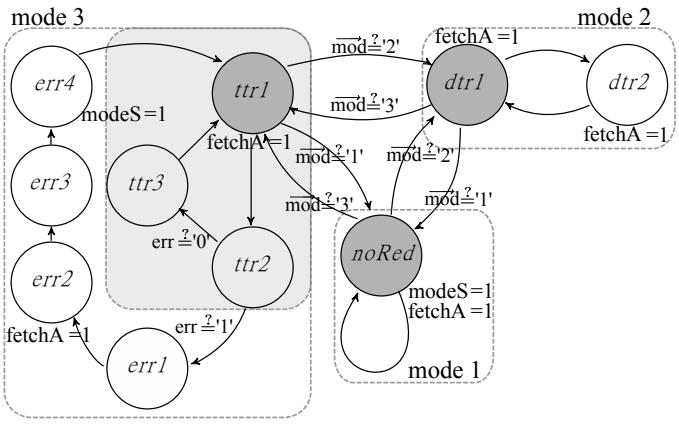

Fig. 5. Control block for DyTR^3.

The labels specify the values of the global control signals (when absent from a node they supposed to be 0). The guard mod is the primary input bus that the environment may use to indicate which mode switch must be performed. In each operating mode, there is only one state allowing a switch. This state ensures that the output stream is consistent (*i.e.,* is not cut in the middle of a redundant series). For example, when switching from mode 3 to 1, the output stream has only triplicated values (when in mode 3) followed by single ones (when in mode 1).

The automaton corresponding to mode 3 has two circular sub-parts: $ttr1 - ttr2 - ttr3$ and $ttr1 - ttr2 - err1 - err2 - err3 - err4$. The former corresponds to the functionality if no soft errors have been detected. The latter is used if the fail signal is raised by *Vot/Detect* during any $3i - 1$ clock cycles indicating a data corruption. The recovery procedure corresponding to this case is described in the next section.

C. Fault tolerance guarantees

In this section, we show that a DyTR^3 circuit in triple-time redundant operating mode (mode 3) is able to mask the effect of any SET within six cycles after its occurrence. In other words, it is fault-tolerant *w.r.t.* the $SET(1, 7)$ fault model. The fault-tolerance properties of mode 2 can be checked with the same reasoning and mode 1 does not have any.

The error-masking properties are again based on the fact that, even if a single SET can corrupt several memory blocks, it can corrupt only one cell in a given memory block. Hereafter, we consider all possible SET occurrence scenarios.

① An SET occurring in the combinational part φ, the signals $\vec{pi}$, $\vec{so}$, $\vec{si}$, or within the *Vot/Detect* may only corrupt d cells (potentially in all memory blocks). Since in mode 3, before an error detection, $modeS = 0$, any SET is logically masked at the multiplexer $MuxB$ and cannot corrupt simultaneously d' and d. Three cases can be distinguished depending on which redundant bit vector is corrupted (*e.g.,* $\vec{e}_1$, $\vec{e}_2$, or $\vec{e}_3$ as in Table 1):

1) If the first redundant bit vector $\vec{e}_1$ is corrupted, then the voting masks the error within three cycles. According to Table 1, the voting is performed on $(\vec{e}_3, \vec{e}_2, \dagger\vec{e}_1)$, next on $(\vec{e}_3, \vec{e}_2, \dagger\vec{e}_1)$, and finally on $(\vec{f}_1, \vec{e}_3, \vec{e}_2)$. In all these votes, $\dagger\vec{e}_1$ is masked by the majority voting.
2) If $\vec{e}_2$ is corrupted, $\dagger\vec{e}_2$ is masked only during the first two votes on $(\vec{e}_3, \dagger\vec{e}_2, \vec{e}_1)$. The third vote $(\vec{f}_1, \vec{e}_3, \dagger\vec{e}_2)$ does not guarantee masking $\dagger\vec{e}_2$. Since the result of the third vote may be incorrect, a corrupted third redundant bit vector propagates. Thus, the error migrates from the 2nd redundant recalculation to the third one. We describe in the third case how an error in the third redundant bit vector is masked.
3) The third redundant bit vector can be corrupted by an SET or, as we just showed, by an error propagation from the second redundant recalculation. In both cases, the *fail* signal will be raised during a $3i - 1$ clock cycle, which indicates that either $\vec{e}_2$ or $\vec{e}_3$ is corrupted. Such case triggers the recovery procedure described below.

The recovery procedure is organized by the DyTR^3 control block (Figure 5). If no error is detected, the control block goes through the $ttr1 - ttr2 - ttr3$ states of its automaton. However, if an error has been detected at a $(3i-1)$ clock cycle (automaton state $ttr2$), then the FSM takes the edge $ttr2 \rightarrow err1$ to start the recovery procedure illustrated in Table 3.

Table 3 presents the detection of a corruption of the third bit vector $\vec{a}_3$ and its recovery. The comparison is done between

d' and d'', so the corrupted vector $\dagger\vec{a}_3$ is detected at cycle 2 (Table 3). At cycle 3, the control block goes to state *err1*. As explained above, cycles 1 and 2 produce a correct result $\vec{a}_v$ at so, but cycle 3 may produce a corrupted bit vector $\dagger\vec{a}_v$. Since the $\dagger\vec{a}_v$ propagates through the combinatorial circuit, the input vector of the sequential part $\dagger\vec{b}_3$ at the same cycle may be corrupted. On the other hand, we know that already computed vectors $\vec{b}_2$ and $\vec{b}_1$ are correct.

At cycle 6, the control block is in state *err4*. It raises the $modeS$ signal, which substitutes the usual vote on (d', d'', s) with a vote on (d', s, s) ignoring the incorrect $\dagger\vec{b}_3$ in d''. As a result, the third redundant bit vector $\vec{c}_3$ is correct and the corrupted $\dagger\vec{b}_3$ disappears from the circuit at cycle 8.

② An SET at the global control wire $fetchA$ can corrupt only one of the inputs of the majority voter. In the worst case, it would corrupt the computation of the third redundant bit vector (during cycles $3i$). Indeed, instead of voting on (d', d'', s), the memory block will vote on (d, d', d''), producing possibly a wrong value. This single error will be detected and corrected as explained in case ①. In the two other cases, the voting produces the correct result because two inputs of the voter remain correct.

③ An SET at the global wire $modeS$ may corrupt the outputs of the multiplexers *MuxB* and *MuxC*. The corruption of *MuxC* substitutes d'' with s or vice versa. However, such substitution alone cannot influence the majority voting at any cycle. During $3i-2$th cycles, the other two redundant bits are correct and, at the other cycles, $d'' = s$. The corruption of *MuxB* is equivalent to a corruption of d'. This has been treated in case ①.

④ Any SET in the centralized control block will be masked within one clock cycle due to its TMR protection.

Other options of SET injection (*e.g.,* inside dynamic delay) lead to the error masking scenarios described above.

V. Dynamic Double-Time Redundancy

Dynamic double-time redundancy can be seen as a simplified version of DyTR^3. Here, we just sketch its main components and features.

A. Memory Block

Each memory cell in the original circuit (with input si and output so) is substituted with the memory block presented in Figure 6. In dynamic double-time redundancy, error masking is fundamentally impossible because there is not enough redundancy. Consequently, the memory blocks perform only error detection using a comparator.

The memory block for DyTR^2 consists of the following components:

- cells d and d' save redundant information for comparison in mode 2; in this operating mode, the input stream is upsampled twice and d and d' contain the same value each odd cycle;

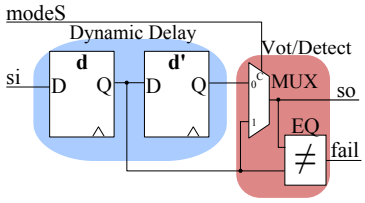

Fig. 6. Memory block for DyTR^2.

- a comparator *EQ* which raises the *fail* signal if d and d' differ; in mode 2, if this signal is raised during an *odd* cycle, it indicates an error detection;
- a multiplexer *MUX* that allows to switch between the double-time redundancy (mode 2 when *modeS*=0) and no redundancy (mode 1 when *modeS*=1).

B. Control block and mode switch

The global control signal *modeS* is set by the control block represented as the FSM in Figure 7.

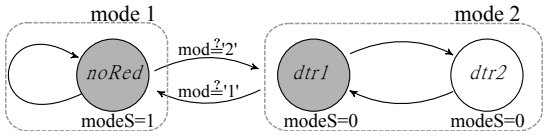

Fig. 7. Control block for DyTR^2.

This FSM is made of two sub-automata, each one represents an operating mode. The single state automaton *notRed* corresponds to the non-redundant mode 1, while the two state automaton $\{dtr1, dtr2\}$ corresponds to mode 2. The label on each state defines the value of the global control signal *modeS* set by the control block. The transition between the two sub-automata models the switches between modes. They depend on the primary input *mod*, which is represented as a guard.

VI. Experimental Results

The proposed transformations have been applied to the *ITC*'99 benchmark suite [8]. We considered three transformations: TMR (with triplicated voters after each memory cell), dynamic double (DyTR^2) and triple (DyTR^3) time redundancy as described before. Each transformed circuit was synthesized for Field-Programmable Gate Array (FPGA) using *Synplify Pro*. We chose the Flash-based ProASIC3 FPGA family as a synthesis target. We protect its data memory with one of the above transformations, whereas its configuration memory is immune to soft errors [9].

Figure 8 illustrates the relative hardware overhead introduced by TMR, DyTR^2, and DyTR^3. The source circuit $b21$ (first bar) consists of a large combinational part (bottom part: 7737 core cells of Flash-based FPGA) and a small sequential part (top part: 490 core cells). In the TMR version of $b21$ (second bar), the triplicated combinational part is dominant. The triplicated voters after each memory cell (for protection against SET) occupy 13.3% of the whole circuit. The DyTR^3 and DyTR^2 circuits (third and fourth bars) reuse the combinational part. For DyTR^3, we explicitly indicated the size of the dynamic delay, the *Vot/Detect* component, and the saving line.

The DyTR^2 circuit has an even smaller area overhead coming from the smaller size of its dynamic delay and the absence of saving line. In DyTR^2 and DyTR^3, the size of the control block is negligible in comparison with the rest of the circuit ($< 1\%$).

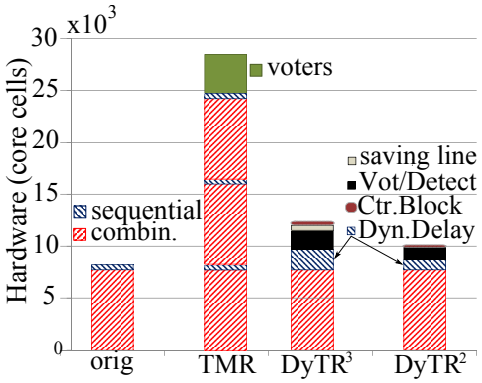

Fig. 8. Transformed circuits profiling (circuit *b21*).

In the following experiments, the circuits of the *ITC*'99 benchmark suite are sorted according to the ratio between the sizes of combinational and sequential parts in the original circuit (written COM/SEQ). Figure 9 shows the circuit size growth (relatively to the original one) after the transformation for highly combinational circuits ($COM/SEQ > 8$, *i.e.,* more than 8 combinational core cells per memory cell).

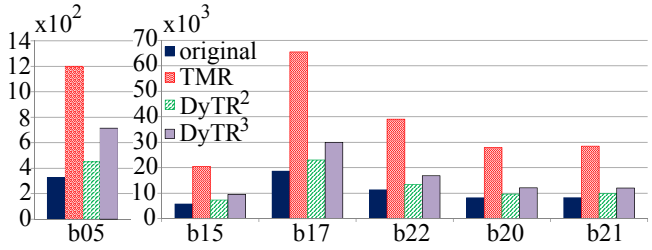

Fig. 9. Circuit size (in core cells) after transformation ($COM/SEQ > 8$).

Figure 9 shows that the DyTR^2 circuits are 1.18 to 1.37 times larger than the original ones, whereas DyTR^3 circuits are 1.46 to 2.17 times larger. For comparison, TMR circuits are 3.4 to 3.65 times larger than the original ones. As a result, DyTR^2 and DyTR^3 circuits are 2.7 to 2.9 and 1.7 to 2.4 smaller than TMR ones.

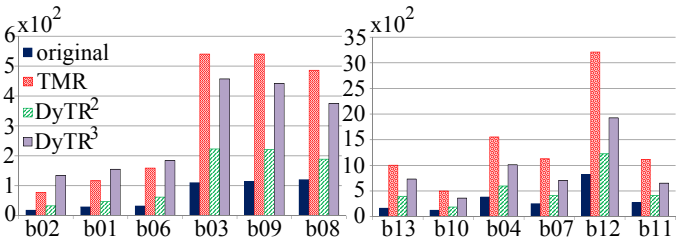

Fig. 10. Circuit size (in core cells) after transformation ($COM/SEQ < 8$).

Figure 10 shows that when the combinational part is small, DyTR^2 and DyTR^3 are still 2.4 to 2.8 and 1.36 to 1.71 smaller on average than TMR. But the attractiveness of time redundancy schemes are lower for circuits that have a small combinational part (*e.g.,* $b02$, $b01$, $b06$, and $b03$). For such circuits, lower hardware benefits and loss in throughput could make the non-adaptive TMR a better option.

The figures do not explicitly represent the overhead of the input/output interface, which are responsible for streams upsampling/dowsampling respectively. Since such interfaces need to be tuned to the surrounding circuit, we do not propose a particular design here. The overall overhead of such interface depends on the number of inputs/outputs wires since a small upsampling/downsampling FSM may have to be inserted for each of them. For several circuits, we have evaluated the overhead to be around 5%.

The DyTR^3 and DyTR^2 fault-tolerance techniques yields adaptive fault-tolerant circuit with a significantly smaller hardware overhead than TMR. As they keep the combinational part unchanged, their hardware benefits compared to TMR grow with the size of the combinational sub-circuit. On the other hand, they decrease the throughput in time-redundant operating modes (> 1). Since a mode n requires an n-upsampling of the input stream, the corresponding throughput is divided by n. This loss is inherent to any time-redundant technique.

We also investigated the relative loss of the maximum synthesizable frequency for the transformed circuits relatively to one of the original circuit. The TMR voters clearly slow down the circuit. The maximum frequency decrease varies from 3-10% for large circuits ($b05$, $b17$, $b20-b22$) to 25-35% for small ones ($b02$, $b03$, $b06$,). In the best case ($b15$, $b21-b22$), the maximum frequency of DyTR^2 circuits is lower than the original one by 1-5%. The control block and the multiplexers in the memory blocks introduce an overhead and the *Vot/Detect* component makes the critical path longer. This is especially visible in circuits with a small combinational part and consequently with low flexibility in combinational optimization ($b13$, $b06$, $b03$). In such cases, the loss in maximum frequency can reach 25-30% which comes close to the loss observed with TMR. A similar behavior is observed for DyTR^3. The *Vot/Detect* circuit is more complex than the one in DyTR^2. The maximum frequency loss is also a bit higher: 1-10% for circuits with a large combinational part ($b15$, $b21 - b22$) and up to 35-44% for small circuits ($b02$, $b06$, $b13$).

VII. Related work

Automatic insertion of fault-tolerance in hardware has been considered mostly for techniques such as TMR and error-correcting codes. The use of time-redundancy is much less standard and usually has an ad-hoc character. It has been used, for instance, to create fault-tolerant designs of arithmetic units [10] or CPUs [11].

Nicolaidis *et al.* [12], [13] presented a related time-redundant Integrated Circuit (IC) transformation. Their fault-tolerant mechanism relies on latching-window masking when an SET glitch is not latched by memory cells since it does not satisfy setup and hold time conditions. A similar technique has been presented in [14]. In comparison with our technique, it requires a strong control of the clock lines. It has been used only with a special hardware support in Application-Specific Integrated Circuit (ASIC) designs and expensive radiation-hardened Xilinx FPGAs.

A specific form of time-redundancy has been widely used in ASIC CPU pipelines [15], [16]. A "shadow" latch is annexed to each original memory cell to implement an error detection mechanism. However, as in the previous cases, the error-detection mechanism cannot be implemented without a dedicated clock line and precise time properties tuning.

The standard synthesis tools for commercial off-the-shelf FPGAs cannot provide the strong clock timing control required by all observed related techniques due to the build-in clock trees. Consequently, such techniques are not applicable for FPGAs, the technology we are mainly aiming for. In addition, previous works consider the effect of sub-critical supply voltages as the main source of soft errors and, as a result, do not take into account all possible fault scenarios (as we do) caused by natural radiation.

Other hardware-based time-redundant techniques rely on check-pointing mechanisms which can be automatically inserted at register-transfer level [17]. The insertion of checkpoints makes such transformation complex while the interface to the surrounding circuit has to be tuned too. We have recently proposed a time-redundant transformation with micro checkpointing and rollback [18] where a kind of dynamic time redundancy is used to speed up the recovery phase and make it transparent for the surrounding circuit. In this paper, we have generalized that principle as a separate technique to produce adaptive fault-tolerant circuits.

VIII. Conclusion

We proposed a novel general logic-level circuit transformation to automatically introduce time-redundancy operating modes for fault-tolerance. As any time-redundant technique, the transformation trades-off the throughput for low hardware overhead. An important novelty is that transformed circuit may dynamically adapt the throughput/fault-tolerance trade-off by changing its operating mode. Therefore, time-redundant modes can be used only in critical situations (*e.g.,* above South Atlantic Anomaly (SAA), Earth poles for satellites), during the processing of crucial data (*e.g.,* encryption of selected data), or critical processes (*e.g.,* satellite computers reboot). When hardware size is limited and fault-tolerance is only occasionally needed, the proposed scheme is a better choice than static TMR, which incurs a constant high area overhead.

After an intuitive presentation of the general dynamic time-redundancy transformation, we have detailed two useful instances (DyTR^2 and DyTR^3) which can detect and/or mask an SET. We have applied the proposed transformations to the $ITC'99$ benchmark suite. The synthesis results for Flash-based FPGA show that DyTR^3 and DyTR^2 circuits are respectively 1.7–2.4 and 2.7–2.9 times smaller than TMR for circuits with big combinatorial part (at least 8 cores cells per memory cell). The hardware overhead of DyTR^3 and DyTR^2 is respectively up to 5.3 and 13.4 times smaller compared to TMR. A comparable loss of maximum synthesizable frequency has been observed for all three circuit transformations (TMR, DyTR^3, DyTR^2).

As in software, time redundancy is only suited to applications that do not always require maximum throughput. A particular target is flash-based FPGA designs (where hardware size is crucial) for embedded systems used in safety critical domains (e.g., physical device controllers, power supply sequencers, crypto cores). Existing FPGA synthesis tools can easily be enriched with our technique. While we focused in this paper on FPGA applications, our technique can also be used for fault-tolerant designs in ASICs.

As future work, we are looking at protecting the clock tree from the effects of SETs and at optimizing the technique with retiming. In addition, we have already proved several related transformations (TMR, simple triple time redundancy and double time redundancy with checkpointing and rollback [18]) using the Cop proof assistant. A formal certification of DyTR would also be needed for complete confidence.

References

[1] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *IEEE Computer*, vol. 38, no. 2, pp. 43–52, Feb. 2005.

[2] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *DSN*, 2002, pp. 389–398.

[3] J. von Neumann, "Probabilistic logic and the synthesis of reliable organisms from unreliable components," *Automata Studies, Princeton Univ. Press*, pp. 43–98, 1956.

[4] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-event upset mitigation selection guide," *Xilinx Application Note XAPP987*, vol. 1, 2008.

[5] A. Sutton, "Creating highly reliable FPGA designs," *Military&Aerospace Technical Bullentin*, vol. Issue 1, pp. 5–7, 2013.

[6] E. J. Daly, J. Lemaire, D. Heynderickx, and D. J. Rodgers, "Problems with models of the radiation belts," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 403–415, Apr 1996.

[7] A. Bogorad *et al.*, "On-orbit error rates of RHBD SRAMs: Comparison of calculation techniques and space environmental models with observed performance," *IEEE Trans. on Nuclear Science*, pp. 2804–2806, 2011.

[8] F. Corno, M. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *Design Test of Computers*, pp. 44–53, 2000.

[9] "Neutron-induced single event upset SEU," *Microsemi Corporation*, no. 55800021-0/8.11, August 2011.

[10] Y.-M. Hsu, V. Piuri, and E. Swartzlander, "Efficient time redundancy for error correcting units and convolvers," in *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, 1995, pp. 198–206.

[11] K. Makoto, A. Masayuki, F. Satoshi, and I. Kazuhiko, "Time redundancy processor with a tolerance to transient faults caused by electromagnetic waves," *DSN*, 2007.

[12] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *17th IEEE VLSI Test Symposium*, 1999, pp. 86–94.

[13] L. Anghel, D. Alexandrescu, and M. Nicolaidis, "Evaluation of a soft error tolerance technique based on time and/or space redundancy," in *13th Symposium on Integrated Circuits and Systems Design*, 2000, pp. 237–242.

[14] F. L. Kastensmidt, C. Luigi, and R. Reis, "Fault-Tolerance Techniques for SRAM-based FPGAs," *Springer, Frontiers in Electronic Testing*, 2006.

[15] D. Ernst *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," in *36th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2003, pp. 7–18.

[16] N. Avirneni, V. Subramanian, and A. Somani, "Low overhead soft error mitigation techniques for high-performance and aggressive systems," in *DSN*, June 2009, pp. 185–194.

[17] C. Chan, D. Schwartz-Narbonne, D. Sethi, and S. Malik, "Specification and synthesis of hardware checkpointing and rollback mechanisms," *Design Automation Conference*, pp. 1222–1228, June 2012.

[18] D. Burlyaev, P. Fradet, and A. Girault, "Automatic time-redundancy transformation for fault-tolerant circuits," *International Symposium on Field-Programmable Gate Arrays*, February 2015.