Automatic Time-Redundancy Transformation for Fault-Tolerant Circuits

Dmitry Burlyaev
Univ. Grenoble Alpes ; INRIA

Pascal Fradet
INRIA ; Univ. Grenoble Alpes

Alain Girault
INRIA ; Univ. Grenoble Alpes

Inria Grenoble
655 avenue de l'Europe, 38330
Montbonnot, France
email: {first}.{last}@inria.fr

ABSTRACT

We present a novel logic-level circuit transformation technique for automatic insertion of fault-tolerance properties. Our transformation uses double-time redundancy coupled with micro-checkpointing, rollback and a speedup mode. To the best of our knowledge, our solution is the only technologically independent scheme capable to correct the multiple bit-flips caused by a Single-Event Transient (SET) with double-time redundancy. The approach allows soft-error masking (within the considered fault-model) and keeps the same input/output behavior regardless error occurrences. Our technique trades-off the circuit throughput for a small hardware overhead. Experimental results on the ITC'99 benchmark suite indicate that the benefits of our methods grow with the combinational size of the circuit. The hardware overhead is 2.7 to 6.1 times smaller than full Triple-Modular Redundancy (TMR) with double loss in throughput. We do not consider configuration memory corruption and our approach is readily applicable to Flash-based FPGAs. Our method does not require any specific hardware support and is an interesting alternative to TMR for logic-intensive designs.

Categories and Subject Descriptors

B.5.3 [**Hardware**]: Register-Transfer-Level Implementation-Reliability and Testing [Redundant design]; B.5.2 [**Hardware**]: Design Aids [Automatic synthesis, Verification]

General Terms

Reliability; Verification

Keywords

Time-Redundancy; Checkpointing; Single-Event Transient; Formal Methods

FPGA'15, February 22–24, 2015, Monterey, California, USA.

http://dx.doi.org/10.1145/2684746.2689058.

1. INTRODUCTION

Circuit tolerance towards soft (non-destructive, non-permanent) errors has become a design characteristic as important as performance and power consumption [1]. Having been an object of attention in space and medical industries for many years [2], circuit fault-tolerance is nowadays a research topic for any application manufactured at modern technology nodes ($90nm$ and smaller) due to the increased risk of soft errors [1]. Such risk results from the continuous shrinking of transistor size that makes components more sensitive to perturbations induced by radiation [3].

The most common methods to make circuits fault-tolerant to soft errors rely on hardware redundancy, therefore incurring a significant area overhead. TMR [4] remains the most popular fault-tolerance technique and is widely supported by CAD tools for FPGAs [5] [6]. Time-redundant techniques for fault-tolerance require less hardware resources than spatial redundancy but decrease the circuit throughput. As in software, the standard implementation relies on a block-by-block processing and triple redundancy: an input data is processed three times to produce three outputs used by a majority voter which filters out a possible error. Consequently, latency and throughput are degraded three times.

In this paper, we propose a circuit transformation that is suitable for any type of processing (block or stream processing) and requires only double time redundancy. We target and evaluate our transformation for flash-based FPGA realization where the small hardware size is especially important and configuration memory upsets are nonexistent [7]. Since the approach does not require specific hardware support, it is applicable to ASICs as well. As any time-redundant scheme, our technique is not suited to applications that require high throughput. A particular target is FPGA-based designs of embedded systems such as controllers used in safety critical domains (space, nuclear, medical, ...).

There are two main types of soft errors caused by particle strikes: Single-Event Upsets (SEUs) (*i.e.,* bit-flips in flip-flops (FFs)) and Single-Event Transients (SETs) (i.e. pulses propagating in the combinational circuit). Since an SET may potentially lead to several bit-flips, SETs subsume SEUs. In this paper, we consider fault models of the form "at most one SET within K clock cycles", denoted by $SET(1, K)$. Even in environments with high levels of ionizing radiations (e.g., space, particle accelerators), K is considered to be larger than 10^{10} [8]. Our transformation masks SETs for any K greater than 10 cycles.

The main features of our Double-Time Redundant Transformation (DTR) transformation are illustrated in Fig. 1. The primary input stream is upsampled twice and given to the combinational part to detect errors by comparison (written C). The line $\ldots, s_1,\ s_2, t_1,\ t_2, \ldots$ represents paired internal states and $\ldots, a,\ a,\ b,\ b, \ldots$ paired bits in the output stream. When an error is detected (*e.g.,* $t_1 \neq t'_2$), a recovery process consisting of a rollback and a re-execution is triggered (resulting in the internal state t_3). The check-pointing mechanism is tolerant towards SETs and is performed every other cycle.

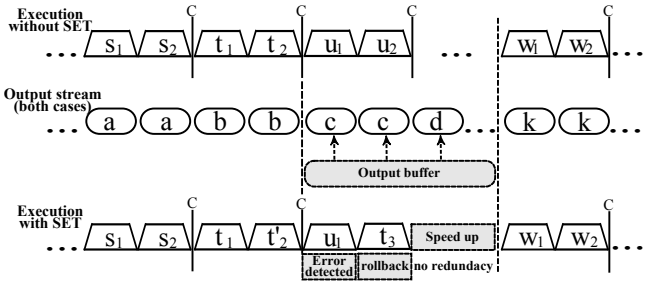

Figure 1: Overview of the DTR transformation

According to the fault-model $SET(1, K)$, no error occurs within K clock cycles after the last error. This allows to switch off time-redundancy during the recovery phase and to "accelerate" the circuit twice (speed up phase in Fig. 1). Along with the use of specifically designed input and output buffers to produce delayed outputs (and record inputs) during that phase, this makes the recovery absolutely transparent to the surrounding circuit. The input/output behavior remains unchanged as if no SET had occurred. The output streams correctness and consistency $(\ldots,\ a,\ a,\ b,\ b, \ldots)$ are guaranteed by the transformation. After an error, the check-pointing mechanism returns the circuit to a correct state (*i.e.,* to the state that the circuit would have been if no error had occurred) within at most *10* clock cycles. Consequently, the allowed maximum fault rate is one every *10* clock cycles (*i.e.,* $SET(1, 10)$).

To summarize, DTR is a new *automatic* logic-level transformation for fault-tolerance with the following benefits:

1. It is technologically independent, in particular it does not require specific hardware support nor control of clock lines. Therefore, it is applicable to Commercial Off-The-Shelf (COTS) FPGAs.

2. Its throughput loss for error-correcting is only double instead of the standard triple overhead.

3. It can be shown to mask all possible SETs, which is a more demanding fault-model than SEUs.

4. It is suitable for stream processing since the input/output streams are insensitive to SET occurrences.

Section 2 introduces notations and provides an overview of the transformation. It consists in replacing each memory cell by a *Memory Block* supporting redundancy and check-pointing, and adding a global *Control Block* providing control signals. Section 3 describes in details the DTR transformation. It ends with the proof that the transformed circuit is fault-tolerant for all possible errors according to the fault-model. Experimental results using the ITC'99 benchmark suite [9] are presented in Section 4. The hardware overheads and maximum throughputs of the original, TMR, and DTR circuits are compared. Section 5 presents related works that use time-redundancy or micro-level checkpointing for circuit fault-tolerance. Finally, we summarize our contributions and sketch possible extensions in Section 6.

2. NOTATIONS AND APPROACH

Any digital circuit can be represented in the most general way as in Figure 2. The circuit, which consists of combinational and sequential parts, takes a primary input bit vector $\vec{PI}$ and returns a primary output bit vector $\vec{PO}$ each clock cycle. The combinational part performs some memoryless boolean function φ.

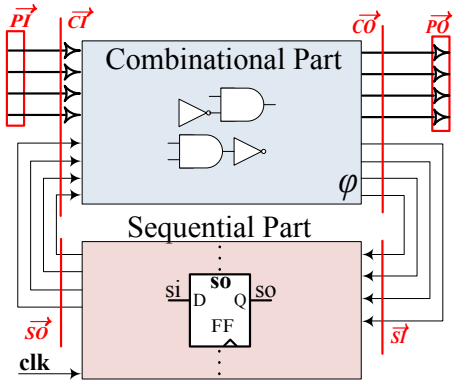

Figure 2: Digital circuit before transformation.

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 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 2 with output *so* is itself denoted *so*.

An SET in a combinational circuit can lead to the non-deterministic corruption of any memory cell connected (by a purely combinational path) to the place where the SET occurred. A corrupted vector is written $\dagger\vec{v}$; it represents the vector $\vec{v}$ with an arbitrary number of bit-flips (corrupted bits). An SET in the combinational circuit of Figure 2 at some cycle i can lead to the corruption of some outputs of the combinational circuit $\dagger\vec{CO}_i$. This leads to the corruption of the primary outputs $\dagger\vec{PO}_i$ and of inputs of the memory cells $\dagger\vec{SI}_i$, which, in turn, causes the corruption of the circuit's memory cells. This last corruption is visible at their outputs during the next clock cycle $\dagger\vec{SO}_{i+1}$. An SET can occur on any input/output wire. Note that SET subsumes the SEU fault-model since any SEU of a cell can be caused by a SET on its input line.

The DTR transformation consists of five steps (see Figure 3):

1. upsampling of the input stream;
2. substitution of each memory cell with a *memory block*;
3. addition of a *control block*;
4. addition of *input buffers* to all circuit primary inputs;
5. addition of *output buffers* to all circuit primary outputs.

Here, the combinational part of the circuit is kept unchanged but $\varphi(\vec{ci})$ is computed twice. The results are compared and, if an error is detected, $\varphi(\vec{ci})$ is recomputed the third time. The input stream is upsampled two times. If $\vec{pi}$ represents the upsampled primary input bit vector of the transformed circuit, it satisfied the following equalities:

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

Each original memory cell is substituted with a memory block that implements the time-redundant mechanism. The memory blocks store the results of signal propagations but they also save recovery bits (or checkpoint bits). As an error-detection mechanism, a comparison takes place that, in case of an error, leads to the use of the recovery bits to rollback and re-execute. The control block takes the result of comparisons as an input and provides several control signals to schedule check-pointing and rollback. To prevent errors from corrupting the input/output behavior, additional input and output buffers are necessary. Input buffers store the last two input vectors to provide the necessary information for re-computation. During the re-computation, the control block speedups the circuit which, in a few cycles, catches up the state it should have had if no error had occurred. Output buffers emit the previously recorded correct outputs and filter out the corrupted data during the recovery process.

In the presented transformations, memory blocks, the control block, and the input/output buffers guarantee that the circuit is fault tolerant, *i.e.,* that an SET (within the fault-model) cannot corrupt the primary outputs. Even errors occurring directly at the primary outputs can be masked. Our output buffers provide three redundant output wires for each output of the original circuit, so the surrounding circuit can also vote to mask errors.

The following sections present the transformation in details.

3. DOUBLE-TIME REDUNDANT TRANSFORMATION

As we observed in Section 2, the circuit after DTR transformation can be represented as in Figure 3. We describe the new components of the DTR transformed circuit (marked with green) hereafter.

3.1 DTR Memory Blocks

The memory block is depicted in Figure 4. It consists of four memory cells:

- two cells d and d' (the data bits) to save redundant information for comparison; since the input stream is upsampled twice, d and d' contain the same value each *odd* cycle; *e.g.,* if the input stream leads to $si_1{=}u$, $si_2{=}u$, ... then the pair (d, d') will contain successively the values $(0,0)$, $(u,0)$, (u,u), ... where the initial values of the memory cells is supposed to be 0;
- two cells r and r' (the recovery bits) with enable-input to keep the value of the si input during four clock cycles and to allow the rollback after an error detection.

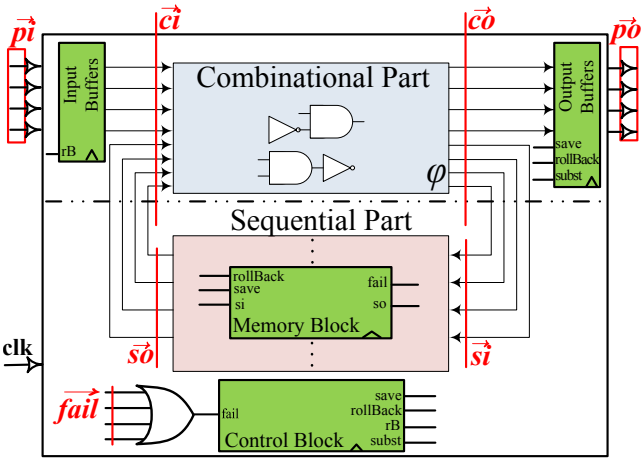

Figure 3: Transformed circuit for DTR.

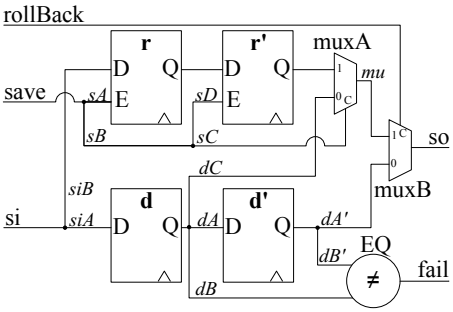

Figure 4: DTR Memory Block.

The DTR memory performs an error-detection comparison whose result is sent as a *fail* signal to the DTR control block. As noted above, the comparison of d and d' is meaningful only during the odd cycles and so is the *fail* signal which is read only at those cycles.

In addition to the data input signal (si in Figure 4), each DTR memory block takes special global control signals *save* and *rollBack* produced by the control block and used to organize the circuit recovery after an error detection.

3.2 DTR Input Buffers

An input buffer is inserted at each primary input of the original circuit to keep the two last bits of the input stream. The buffer is implemented as a pipeline of two memory cells, b and b' as shown in Figure 5. The signal rB is raised by the control block during the recovery process (Figure 7).

The cells b and b' are used only during the recovery process in order to re-execute the last two cycles. These bits are provided to the combinational part instead of the bits from the input streams. They also serve to store the inputs that

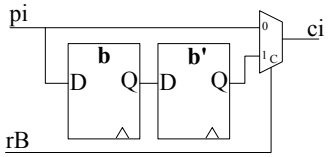

Figure 5: DTR input buffer (pi primary input)

keep coming during those two cycles. During the recovery, the vector $\vec{ci}$ consists of (i) the first part $\vec{pi}$ coming from the input buffers and (ii) the second part $\vec{so}$ coming from the rollbacked memory blocks. If the error is detected at cycle i, then the rollback is performed at cycle $i+1$ and the vector $\vec{pi}_{i-1} \oplus \vec{so}_{i-1}$ is provided to the combinational part (exactly the input vector already supplied 2 cycles before).

From Eq. (2), we see that b and b' represent two identical (resp. distinct) upsampled bits at each odd (resp. even) clock cycle: $\vec{b}_{2i-1} = \vec{b'}_{2i-1}$. Since the error detection occurs at odd cycles, the recovery, which starts a cycle after, will read two different inputs (*i.e.,* not the same upsampled input) from $\vec{b}$ and $\vec{b'}$. This is consistent with the speedup of the circuit during recovery. The behavior of input buffers during recovery is illustrated in Section 3.6.

3.3 DTR Output Buffers

The error recovery procedure disturbs the vector stream $\vec{co}$ in comparison with the normal operating mode. To mask this effect at the primary outputs, we insert a DTR output buffer (Figure 6) before each primary output. They produce correct outputs but introduce in normal mode a latency loss of three clock cycles.

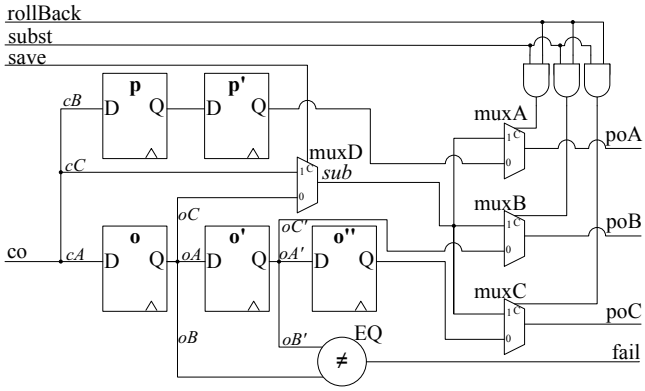

Figure 6: DTR Output Buffer (co output of the combinational part.

The buffer is designed to be also fault-tolerant to any SET occurring inside or at its outputs. To achieve this property, the new primary outputs are triplicated (poA, poB, poC). The output buffers ensure that at least two out of them are correct at each *even* cycle. The surrounding circuit can thus read these outputs at even cycles and perform a vote to mask any SET that may have occurred at the outputs. This is just a possible implementation and a different design could be used, *e.g.,* with a fault-model excluding/disregarding errors at the outputs or with different interface requirements.

Additional details about the behavior of output buffers are provided in Section 3.6.

3.4 DTR Control Block

The control block is shown in Figure 3. The control signals *save*, *rollBack* (for memory blocks), *rB* (for input buffers), and *subst* (for output buffers) are generated to support the transformed circuit functionality during *normal* and *recovery* modes.

Control block takes as an input the error detection signal *fail* (the disjunction of all memory blocks and output buffers individual *fail* signals). The functionality of the control block can be described as the Finite State Machine (FSM) of Figure 7. States *norm1* and *norm2* compose the *normal* mode which raises the *save* signal alternatively. When an error is detected (*i.e., fail* = 1), the FSM enters the recovery mode for 4 cycles and raises the corresponding signals.

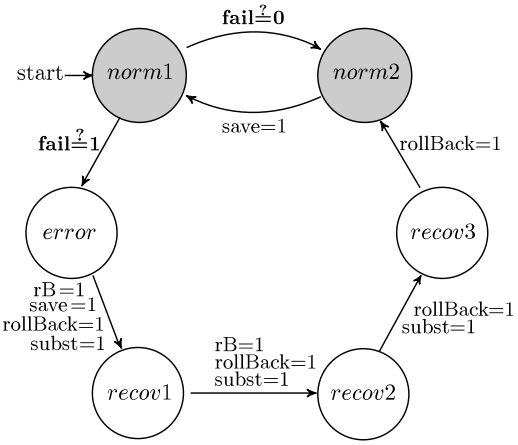

Figure 7: FSM of the DTR control block: "$\stackrel{?}{=}$" denotes a guard, "=" an assignment and signals absent from an edge are set to 0.

The control block itself is protected against SETs using TMR. Since its size (25 core cells) is negligible in comparison with the rest of the circuit, its triplication almost does not increase the hardware overhead (as confirmed by Figure 10). Therefore, the only way to corrupt the global control signals is by an SET outside the control block. This ensures that no two global control signals can be corrupted simultaneously.

It would be tedious to explain separately all the possible interactions of the control block with memory blocks and buffers. Instead, we present in Sections 3.5 and 3.6 the two operating modes of the DTR circuit: the *normal* mode (before a soft-error) and the *recovery* mode (after a soft-error). Section 3.7, which examines all possible SETs, also clarifies the mechanisms of the different components.

3.5 Normal Execution Mode

If no error is detected, the circuit is working in the normal operating mode. During this mode, the signal *rollBack* is always set to zero, while *save* is raised every even cycle:

$$save_{2i-1} = 0 \text{ and } save_{2i} = 1 \tag{3}$$

Since *save* is the enable signal of r and r', it organizes a four-cycle delay from si to r' in normal mode. The internal behavior of each DTR memory block in normal mode can

be described by the following equations:

$$\begin{cases} rollBack_i = 0 \\ \vec{si}_i = \vec{d}_{i+1} = \vec{d'}_{i+2} = \vec{so}_{i+2} \\ \vec{si}_{2i} = \vec{r}_{2i+1} = \vec{r}_{2i+2} = \vec{r'}_{2i+3} = \vec{r'}_{2i+4} \\ save_{2i-1} = 0,\ save_{2i} = 1 \end{cases} \quad (4)$$

It is easy to show that the DTR circuit verifies the same equalities as Eq. (1) for the original circuit:

$$\vec{co}_i = \varphi(\vec{ci}_i) \qquad \vec{ci}_i = \vec{pi}_i \oplus \vec{so}_i \qquad \vec{co}_i = \vec{po}_i \oplus \vec{si}_i \quad (5)$$

From Eqs. (2), (4), and (5), we can derive two properties for the normal operating mode. First, the output bit stream $\vec{co}$ of the combinational part after the circuit transformation is a double-time upsampling of the corresponding bit stream $\vec{CO}$ of the original circuit. Formally:

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

PROOF. We assume that the two cells d and d' of each memory block are initialized as the original cell, and therefore $\vec{so}_1 = \vec{so}_2 = \vec{SO}_1$. By Eqs. (1) and (5), we have $\vec{co}_1 = \vec{co}_2 = \vec{CO}_1$. The proof is then a simple induction using Eqs. (1), (2), and (5). □

Second, at each odd cycle, the outputs of the cells d and d' are equal.

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

PROOF. At the first cycle ($i=1$), the property is true by the same initialization hypothesis as above. Property 2 and Eq. (5) entail that $\vec{si}_{2i-1} = \vec{si}_{2i}$. By Eq. (4), we have:

$$\begin{array}{ccccc} \vec{si}_{2i} & = & \vec{d}_{2i+1} & = & \vec{d'}_{2i+2} \\ \| & & & & \\ \vec{si}_{2i-1} & = & \vec{d}_{2i} & = & \vec{d'}_{2i+1} \end{array}$$

and thus, $\forall i > 0, \vec{d}_{2i+1} = \vec{d'}_{2i+1}$, which is equivalent to $\forall i > 1,\ \vec{d}_{2i-1} = \vec{d'}_{2i-1}$. □

For error detection, we check the violation of Property 2 which is performed by the *EQ* comparator (Figure 4). If at some odd cycle $2j-1$ the d and d' cells of a memory block differ, an error is detected and the *fail* signal will be raised ($fail_{2j-1} = 1$). The circuit has to rollback to the correct state stored in $\vec{r'}$ and to re-compute the previous step. The rollback is performed by propagating $\vec{r'}$ to $\vec{so}$. From Eq. (4), we can derive the following equation:

$$\vec{r'}_{2j-1} = \vec{r'}_{2j} = \vec{si}_{2j-4} \quad (6)$$

Eq. (6) means that, at the moment of an error detection (and at the next clock cycle), the recovery bit r' is set to the value of the input signal si 3 cycles before. It will be shown in Section 3.7 that all recovery bits contain correct values when an error is detected (*i.e.,* an error in the data bits never corrupts $\vec{r'}$).

3.6 Recovery Execution Mode

If an error has been detected, the circuit performs a rollback followed by three consecutive cycles during which the double time redundancy mechanism is switched-off. These steps are implemented by a sequence of signals (*save*, *rollBack*, *subst*, and *rB*) produced by the control block.

The left part of Table 3.5 (in white) shows the values of bit vectors in the transformed circuit cycle by cycle when an error is detected at clock cycle i. The behavior of the circuit in normal mode (when no error occurs) is shown in the right part (in gray). Recall that, in the normal mode, the vector $\vec{ci}$ at cycle i is such that $\vec{ci}_i = \vec{pi}_i \oplus \vec{so}_i = \vec{pi}_i \oplus \vec{si}_{i-2}$. The principle of the rollback mechanism is that the DTR memory blocks re-inject the last correct saved state (the $\vec{si}$ vector) while the DTR input buffers will re-inject the corresponding primary input (the $\vec{pi}$ component).

At the clock cycle $(i+1)$ following an error detection, the recovery starts and the correct state represented by $\vec{r'}$ is pushed through $\vec{so}$. Consequently, $\vec{so}_{i+1} = \vec{r'}_{i+1} = \vec{si}_{i-3}$ instead of the expected $\vec{si}_{i-1}$ in the normal mode. Thus, the second component of $\vec{ci}_{i+1}$ is $\vec{si}_{i-3}$. The primary input vector is also replaced by the vector kept in the input buffers; that is, at the $i+1$ cycle $\vec{pi}_{i+1}$ is replaced by $\vec{pi}_{i-1}$. Recall that, during recovery, the circuit is working with the throughput of the original circuit, which is twice higher than in the normal mode. In particular, during the cycles $i+2$, $i+3$, and $i+4$, $\vec{d}$ propagates directly through the $\vec{so}$ outputs of each memory block, bypassing the memory cells $\vec{d'}$. This is implemented by raising *rollBack* and lowering *save* which control the *muxA* and *muxB* multiplexers appropriately in each memory block. This is safe since the $SET(1, K)$ fault-model guarantees that no additional error can occur just after a SET.

Consider cycle $i+2$: the second component of $\vec{ci}_{i+2}$ is $\vec{si}_{i-1}$ ($\vec{si}_{i-2}$, which is identical to $\vec{si}_{i-1}$, has been skipped). Similarly, the primary input vector is replaced by $\vec{pi}_{i+1}$, since, in the input buffers, $\vec{b'}_{i+2} = \vec{pi}_i$ and $\vec{pi}_{i+1} = \vec{pi}_i$. It follows that $\vec{ci}_{i+1} = \vec{pi}_{i-1} \oplus \vec{si}_{i-3}$ and $\vec{ci}_{i+2} = \vec{pi}_{i+1} \oplus \vec{si}_{i-1}$.

Let us look more closely at how an error propagates and how it is masked. The error $\vec{d}_i \neq \vec{d'}_i$ detected at cycle i does not indicate which of $\vec{d}$ or $\vec{d'}$ is corrupted. The fault-model only guarantees that their simultaneous corruption is not possible. We consider both of them as potentially corrupted and the † and ‡ marks indicate the two possible bit vector corruptions. We track the error propagation cycle by cycle based on data dependencies between vectors as shown in Table 3.5.

Case #1: If $\vec{d'}_i$ contains a corrupted value $\dagger\vec{si}_{i-2}$, it contaminates $\vec{ci}_i$. Since this input bit vector is corrupted, the outputs of the combinational circuit can be corrupted as well as $\vec{d}_{i+1}$ that latches $\dagger\vec{si}_i$. This corrupted value propagates to $\vec{d'}$, so $\vec{d'}_{i+2} = \dagger\vec{si}_i$. Since $\vec{d'}$ is bypassed and $\vec{d}$ propagates directly through the wires $\vec{so}$, the error at $\dagger\vec{si}_i$ is logically masked at *muxB* by *rollBack*, which is raised during 4 cycles after the error detection.

Case #2: If $\vec{d}_i$ contains a corrupted value $\ddagger\vec{si}_{i-1}$, it will propagate to $\vec{d'}$ and $\vec{d'}_{i+1} = \ddagger\vec{si}_{i-1}$. Since $\ddagger\vec{si}_{i-1}$ has been latched by $\vec{d}$ and $\vec{r}$ at the same clock cycle, $\vec{r}_i$ is also corrupted: $\vec{r}_i = \ddagger\vec{si}_{i-1}$. When rollback happens at cycle $i+1$, $\vec{r}$ propagates to $\vec{r'}$ and remains in $\vec{r'}$ until the next raised *save*. The *save* signal is raised only 5 cycles after the error-detection, when *rollBack* is lowered again. As a result, any error in $\vec{r'}_{i+5}$ will be re-written with a new correct data and cannot propagate through signals $\vec{so}$ due to the logical masking by $rollBack = 0$ at *muxB*.

All corrupted signals disappear from the circuit state within 6 clock cycles after an error detection. The whole circuit returns to a correct state within 8 cycles. As it is shown in

Table 3.5: Recovery process in DTR circuits.

clk	$\vec{pi}$	$\vec{b}$	$\vec{b'}$	$\vec{ci}$	$\vec{d}$	$\vec{d'}$	$\vec{r}$	$\vec{r}'$	f	sa	ro	$\vec{ci}$	$\vec{d}$	$\vec{d'}$	$\vec{r}$	$\vec{r}'$
$i-3$	$\vec{pi}_{i-3}$	$\vec{pi}_{i-4}$	$\vec{pi}_{i-5}$	$\vec{pi}_{i-3} \oplus \vec{si}_{i-5}$	$\vec{si}_{i-4}$	$\vec{si}_{i-5}$	$\vec{si}_{i-5}$	$\vec{si}_{i-7}$	?	1	0	$\vec{pi}_{i-3} \oplus \vec{si}_{i-5}$	$\vec{si}_{i-4}$	$\vec{si}_{i-5}$	$\vec{si}_{i-5}$	$\vec{si}_{i-7}$
$i-2$	$\vec{pi}_{i-2}$	$\vec{pi}_{i-3}$	$\vec{pi}_{i-4}$	$\vec{pi}_{i-2} \oplus \vec{si}_{i-4}$	$\vec{si}_{i-3}$	$\vec{si}_{i-4}$	$\vec{si}_{i-3}$	$\vec{si}_{i-5}$	0	0	0	$\vec{pi}_{i-2} \oplus \vec{si}_{i-4}$	$\vec{si}_{i-3}$	$\vec{si}_{i-4}$	$\vec{si}_{i-3}$	$\vec{si}_{i-5}$
$i-1$	$\vec{pi}_{i-1}$	$\vec{pi}_{i-2}$	$\vec{pi}_{i-3}$	$\vec{pi}_{i-1} \oplus \vec{si}_{i-3}$	$\vec{si}_{i-2}$	$\vec{si}_{i-3}$	$\vec{si}_{i-3}$	$\vec{si}_{i-5}$	?	1	0	$\vec{pi}_{i-1} \oplus \vec{si}_{i-3}$	$\vec{si}_{i-2}$	$\vec{si}_{i-3}$	$\vec{si}_{i-3}$	$\vec{si}_{i-5}$
$\boldsymbol{i}$	$\vec{pi}_{i}$	$\vec{pi}_{i-1}$	$\vec{pi}_{i-2}$	$\vec{pi}_{i} \oplus \dagger\vec{si}_{i-2}$	$\ddagger\vec{si}_{i-1}$	$\dagger\vec{si}_{i-2}$	$\ddagger\vec{si}_{i-1}$	$\vec{si}_{i-3}$	1	0	0	$\vec{pi}_{i} \oplus \vec{si}_{i-2}$	$\vec{si}_{i-1}$	$\vec{si}_{i-2}$	$\vec{si}_{i-1}$	$\vec{si}_{i-3}$
$i+1$	$\vec{pi}_{i+1}$	$\vec{pi}_{i}$	$\vec{pi}_{i-1}$	$\vec{pi}_{i-1} \oplus \vec{si}_{i-3}$	$\dagger\vec{si}_{i}$	$\ddagger\vec{si}_{i-1}$	$\ddagger\vec{si}_{i-1}$	$\vec{si}_{i-3}$	?	1	1	$\vec{pi}_{i+1} \oplus \vec{si}_{i-1}$	$\vec{si}_{i}$	$\vec{si}_{i-1}$	$\vec{si}_{i-1}$	$\vec{si}_{i-3}$
$i+2$	$\vec{pi}_{i+2}$	$\vec{pi}_{i+1}$	$\vec{pi}_{i}$	$\vec{pi}_{i+1} \oplus \vec{si}_{i-1}$	$\vec{si}_{i-1}$	$\dagger\vec{si}_{i}$	$\vec{si}_{i-1}$	$\ddagger\vec{si}_{i-1}$	?	0	1	$\vec{pi}_{i+2} \oplus \vec{si}_{i}$	$\vec{si}_{i+1}$	$\vec{si}_{i}$	$\vec{si}_{i+1}$	$\vec{si}_{i-1}$
$i+3$	$\vec{pi}_{i+3}$	$\vec{pi}_{i+2}$	$\vec{pi}_{i+1}$	$\vec{pi}_{i+3} \oplus \vec{si}_{i+1}$	$\vec{si}_{i+1}$	$\vec{si}_{i-1}$	$\vec{si}_{i-1}$	$\ddagger\vec{si}_{i-1}$	?	0	1	$\vec{pi}_{i+3} \oplus \vec{si}_{i+1}$	$\vec{si}_{i+2}$	$\vec{si}_{i+1}$	$\vec{si}_{i+1}$	$\vec{si}_{i-1}$
$i+4$	$\vec{pi}_{i+4}$	$\vec{pi}_{i+3}$	$\vec{pi}_{i+2}$	$\vec{pi}_{i+4} \oplus \vec{si}_{i+3}$	$\vec{si}_{i+3}$	$\vec{si}_{i+1}$	$\vec{si}_{i-1}$	$\ddagger\vec{si}_{i-1}$	?	0	1	$\vec{pi}_{i+4} \oplus \vec{si}_{i+2}$	$\vec{si}_{i+3}$	$\vec{si}_{i+2}$	$\vec{si}_{i+3}$	$\vec{si}_{i+1}$
$i+5$	$\vec{pi}_{i+5}$	$\vec{pi}_{i+4}$	$\vec{pi}_{i+3}$	$\vec{pi}_{i+5} \oplus \vec{si}_{i+3}$	$\vec{si}_{i+4}$	$\vec{si}_{i+3}$	$\vec{si}_{i-1}$	$\ddagger\vec{si}_{i-1}$	?	1	0	$\vec{pi}_{i+5} \oplus \vec{si}_{i+3}$	$\vec{si}_{i+4}$	$\vec{si}_{i+3}$	$\vec{si}_{i+3}$	$\vec{si}_{i+1}$
$i+6$	$\vec{pi}_{i+6}$	$\vec{pi}_{i+5}$	$\vec{pi}_{i+4}$	$\vec{pi}_{i+6} \oplus \vec{si}_{i+4}$	$\vec{si}_{i+5}$	$\vec{si}_{i+4}$	$\vec{si}_{i+5}$	$\vec{si}_{i-1}$	0	0	0	$\vec{pi}_{i+6} \oplus \vec{si}_{i+4}$	$\vec{si}_{i+5}$	$\vec{si}_{i+4}$	$\vec{si}_{i+5}$	$\vec{si}_{i+3}$
$i+7$	$\vec{pi}_{i+7}$	$\vec{pi}_{i+6}$	$\vec{pi}_{i+5}$	$\vec{pi}_{i+7} \oplus \vec{si}_{i+5}$	$\vec{si}_{i+6}$	$\vec{si}_{i+5}$	$\vec{si}_{i+5}$	$\vec{si}_{i-1}$	?	1	0	$\vec{pi}_{i+7} \oplus \vec{si}_{i+5}$	$\vec{si}_{i+6}$	$\vec{si}_{i+5}$	$\vec{si}_{i+5}$	$\vec{si}_{i+3}$
$i+8$	$\vec{pi}_{i+8}$	$\vec{pi}_{i+7}$	$\vec{pi}_{i+6}$	$\vec{pi}_{i+8} \oplus \vec{si}_{i+6}$	$\vec{si}_{i+7}$	$\vec{si}_{i+6}$	$\vec{si}_{i+7}$	$\vec{si}_{i+5}$	0	0	0	$\vec{pi}_{i+8} \oplus \vec{si}_{i+6}$	$\vec{si}_{i+7}$	$\vec{si}_{i+6}$	$\vec{si}_{i+7}$	$\vec{si}_{i+5}$

$\dagger = \ddagger$ but for two mutually-exclusive error propagation cases
f:fail; sa:save; ro:rollBack

the next section, the error detection occurs at worst 2 cycles later after an SET.

Table 3.6 represents the behavior of output buffers in the same situation (*i.e.,* the recovery procedure when an error is detected at cycle i). The signal names correspond to Figure 6.

We investigate all possible SETs in the next section.

3.7 Fault Tolerance Guarantees

Hereafter, we check all possible SET insertion cases. We write j to denote the clock cycle where the SET occurs. The causal relationship is written as "$\rightarrow$".

① An SET in $\vec{ci}$, $\vec{si}$, the *rollBack* signal, the internal wire dA', or the combinational part φ may lead to $\dagger\vec{d}$ and $\dagger\vec{r}$. During odd cycles, the simultaneous corruption of $\vec{d}$ and $\vec{r}$ is not possible since the *save* signal logically masks SET propagation towards r memory cell. As a result, there are two cases:

1. $\dagger\vec{d}_{j+1}$; *if* $j = 2i-1$. If $\vec{d}$ has been corrupted by an SET in the preceding combinational circuit, an error will be detected by the comparator within the next two cycles. $fail_{j+2} = 1$ since $\vec{d}_{j+2}$ is calculated correctly. Since $\vec{r'}$ is correct, the recovery will return the circuit to its correct state.

2. $\dagger\vec{d}_{j+1} \wedge \dagger\vec{r}_{j+1}$; *if* $j = 2i$. In this case, we must check that the error is detected before reaching $\vec{r'}$. Actually, the error will be detected at the next (odd) clock cycle after an error occurrence: $fail_{j+1} = 1$. But $\vec{r'}$ keeps its correct value because $save_{j+1} = 0$. The recovery process starts at cycle $j+2$, re-writing the correct $\vec{r'}$ with a possible corrupted data, but in the same cycle $\vec{r'}_{j+2}$ outputs a correct value that rollbacks the circuit to a correct state.

② Consider the following SETs: $\dagger save_j$, $\dagger\vec{r}_j$, $\dagger\vec{r'}_j$, $\dagger\ \vec{mu}_j$, $\dagger\ siB_j$ and $\dagger\vec{dC}_j$, which may result in the corruption of the pipeline $r - r'$ (see Figure 4). Such corruption is not detected by the comparator and cannot propagate to the data bits. It disappears a few cycles later at *muxB* by *rollBack* = 0. So, this failure is masked.

③ An SET during an odd clock cycle at the *fail* line possibly leads to spurious error detection followed by a recovery. But $\vec{r'}_{j+1}$ is valid and the recovery will be performed correctly. During even clock cycles, an SET at the *fail* line remains silent since, at these cycles, a *fail* is ignored by the control block.

④ An SET at the output signal of d' may lead to three different cases:

1. $\dagger\vec{d'}_j \rightarrow \dagger\vec{so}_j \rightarrow \dagger\varphi_j$, which is equivalent to case ①;
2. $\dagger\vec{d'}_j \rightarrow \dagger\vec{fail}_j$, which is equivalent to case ③;
3. $\dagger\vec{d'}_j \rightarrow \dagger\varphi_j \wedge \dagger\vec{fail}_j$ *i.e.,* a simultaneous corruption of combinational circuit and a *fail* signal. The recovery process starts at the next clock cycle $j+1$ using the correct $\vec{r'}_{j+1}$.

⑤ An SET at the output signal of d may lead to the corruption of dA, dB, and/or dC (see Figure 4). First, a corruption of dC will always be masked regardless of the possible common corruptions of dA or dB. Indeed, if dC is corrupted, then the propagation/corruption will be masked by *muxB* since *rollBack* = 0 (a simultaneous corruption is impossible). Five other cases must be considered:

- If the error propagates to dA (but not dB) and is latched by d' during an *even* cycle, then an error will be detected at the next odd cycle $j+1$ and will be masked as in case ①;
- If the error propagates to dA (but not dB) and is latched by d' during an *odd* cycle, then it is equivalent to a corruption of the combinational circuit one clock cycle after the latch. It will be masked as in case ①;
- If the error propagates to dA and dB and corrupts d' and *fail* at an even cycle, then we are back to the first case above; indeed, the control block reads *fail* only at odd cycles and the corruption of *fail* will not be considered;

Table 3.6: Recovery Process: Input/Output Buffers Reaction for an Error Detection at Cycle i

clk	$\vec{pi}$	$\vec{ci}$	$\vec{o}$	$\vec{o}'$	$\vec{o}''$	$\vec{poA/B/C}$	$fail$	sav	ro	rB	sub	$\vec{o}$	$\vec{o}'$	$\vec{o}''$	$\vec{poA/B/C}$
$i-3$	$\vec{pi}_{i-3}$	$\vec{pi}_{i-3} \oplus \vec{si}_{i-5}$	$\vec{co}_{i-4}$	$\vec{co}_{i-5}$	$\vec{co}_{i-6}$	$\vec{co}_{i-5}$	?	1	0	0	0	$\vec{co}_{i-4}$	$\vec{co}_{i-5}$	$\vec{co}_{i-6}$	$\vec{co}_{i-5} = \vec{co}_{i-6}$
$i-2$	$\vec{pi}_{i-2}$	$\vec{pi}_{i-2} \oplus \vec{si}_{i-4}$	$\vec{co}_{i-3}$	$\vec{co}_{i-4}$	$\vec{co}_{i-5}$	*ignore*	0	0	0	0	0	$\vec{co}_{i-3}$	$\vec{co}_{i-4}$	$\vec{co}_{i-5}$	*ignore*
$i-1$	$\vec{pi}_{i-1}$	$\vec{pi}_{i-1} \oplus \vec{si}_{i-3}$	$\vec{co}_{i-2}$	$\vec{co}_{i-3}$	$\vec{co}_{i-4}$	$\vec{co}_{i-3}$	?	1	0	0	0	$\vec{co}_{i-2}$	$\vec{co}_{i-3}$	$\vec{co}_{i-4}$	$\vec{co}_{i-3} = \vec{co}_{i-4}$
i	$\vec{pi}_i$	$\vec{pi}_i \oplus \dagger\vec{si}_{i-2}$	$\ddagger\vec{co}_{i-1}$	$\ddagger\vec{co}_{i-2}$	$\vec{co}_{i-3}$	*ignore*	1	0	0	0	0	$\vec{co}_{i-1}$	$\vec{co}_{i-2}$	$\vec{co}_{i-3}$	*ignore*
$i+1$	$\vec{pi}_{i+1}$	$\vec{pi}_{i-1} \oplus \vec{si}_{i-3}$	$\dagger\vec{co}_i$	$\ddagger\vec{co}_{i-1}$	$\ddagger\vec{co}_{i-2}$	$\vec{co}_{i-1}$ (←)	?	1	1	1	1	$\vec{co}_i$	$\vec{co}_{i-1}$	$\vec{co}_{i-2}$	$\vec{co}_{i-1} = \vec{co}_{i-2}$
$i+2$	$\vec{pi}_{i+2}$	$\vec{pi}_{i+1} \oplus \vec{si}_{i-1}$	$\vec{co}_{i-1}$	$\dagger\vec{co}_i$	$\ddagger\vec{co}_{i-1}$	*ignore*	?	0	1	1	1	$\vec{co}_{i+1}$	$\vec{co}_i$	$\vec{co}_{i-1}$	*ignore*
$i+3$	$\vec{pi}_{i+3}$	$\vec{pi}_{i+3} \oplus \vec{si}_{i+1}$	$\vec{co}_{i+1}$	$\vec{co}_{i-1}$	$\dagger\vec{co}_i$	$\vec{co}_{i+1}$ (←)	?	0	1	0	1	$\vec{co}_{i+2}$	$\vec{co}_{i+1}$	$\vec{co}_i$	$\vec{co}_{i+1} = \vec{co}_i$
$i+4$	$\vec{pi}_{i+4}$	$\vec{pi}_{i+4} \oplus \vec{si}_{i+3}$	$\vec{co}_{i+3}$	$\vec{co}_{i+1}$	$\vec{co}_{i-1}$	*ignore*	?	0	1	0	0	$\vec{co}_{i+3}$	$\vec{co}_{i+2}$	$\vec{co}_{i+1}$	*ignore*
$i+5$	$\vec{pi}_{i+5}$	$\vec{pi}_{i+5} \oplus \vec{si}_{i+3}$	$\vec{co}_{i+4}$	$\vec{co}_{i+3}$	$\vec{co}_{i+1}$	$\vec{co}_{i+3}$	?	1	0	0	0	$\vec{co}_{i+4}$	$\vec{co}_{i+3}$	$\vec{co}_{i+2}$	$\vec{co}_{i+3} = \vec{co}_{i+2}$
$i+6$	$\vec{pi}_{i+6}$	$\vec{pi}_{i+6} \oplus \vec{si}_{i+4}$	$\vec{co}_{i+5}$	$\vec{co}_{i+4}$	$\vec{co}_{i+3}$	*ignore*	0	0	0	0	0	$\vec{co}_{i+5}$	$\vec{co}_{i+4}$	$\vec{co}_{i+3}$	*ignore*
$i+7$	$\vec{pi}_{i+7}$	$\vec{pi}_{i+7} \oplus \vec{si}_{i+5}$	$\vec{co}_{i+6}$	$\vec{co}_{i+5}$	$\vec{co}_{i+4}$	$\vec{co}_{i+5}$	?	1	0	0	0	$\vec{co}_{i+6}$	$\vec{co}_{i+5}$	$\vec{co}_{i+4}$	$\vec{co}_{i+5} = \vec{co}_{i+4}$
$i+8$	$\vec{pi}_{i+8}$	$\vec{pi}_{i+8} \oplus \vec{si}_{i+6}$	$\vec{co}_{i+7}$	$\vec{co}_{i+6}$	$\vec{co}_{i+5}$	*ignore*	0	0	0	0	0	$\vec{co}_{i+7}$	$\vec{co}_{i+6}$	$\vec{co}_{i+5}$	*ignore*

‡ = † but for two error-detection cases: ‡ - detection in Output Buffer; † - detection in a preceding Memory Block
(←) - data substitution performed by multiplixers $muxA$, $muxB$, $muxC$, $muxD$
sav - save; ro - rollBack; sub - subst

- If the error propagates to dA and dB and corrupts d' and *fail* at an odd cycle, then the recovery starts at the next clock cycle using the correct $\vec{r'}$ and disregarding the corrupted $\vec{d'}$;
- If the error propagates to dB (but not dA), it may corrupt the *fail* signal and it is masked as in case ③.

An SET in the control block is fixed within one clock cycle since it is protected by TMR. The hardware overhead of TMR for this modest sub-circuit is small.

An SET may also occur in input buffers, in particular in the memory cells b and b' (see Figure 5). Such an error will be logically masked within two clock cycles by the signal $rB=0$ at the multiplexer.

The most critical SET is the one that may occur at the primary input signals $\vec{pi}$ and is latched both by the memory blocks and the input buffers. Double-redundancy can detect the error but it has no way to replay the input signal. Such an SET can be masked only if a surrounding circuit can read the *fail* signal from the DTR circuit and provide a third copy of $\vec{pi}$. Here, we do not enforce such a requirement on the surrounding circuit and consider that the fault-model forbids the corruption of the primary inputs.

An SET occurring just before an output buffer at $\vec{co}$ (see Figure 6) will be detected by the comparator (like in memory blocks). The error will be masked at multiplexers $muxA$, $muxB$, or $muxC$. The structure of the output buffer provides an isolation for the pipelines $o-o'-o''$ and $p-p'$, which in turn guarantees that at least two memory cells among o', o'', and p' are correct during all even clock cycles. The new primary outputs, poA, poB, and poC, are identical during all even clock cycles if no SET occurs, and only one can differ from the others if an SET occurs. This fault-tolerance property still holds even if one of the control signals (*rollBack*, *subst*, or *save*) is corrupted by an SET. Furthermore, using three outputs, as in TMR, gives the surrounding circuit the capability to mask (by voting) any error occurring at the primary outputs.

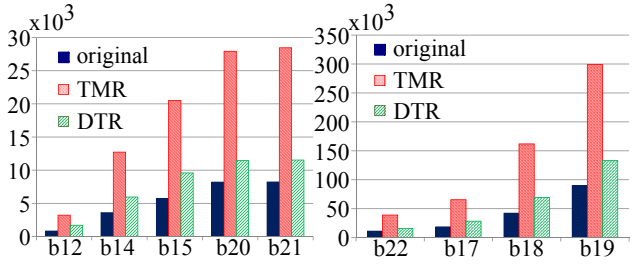

Figure 8: Circuit size (in core cells) after transformation (large circuits).

4. EXPERIMENTAL RESULTS

The proposed DTR transformation has been applied to the *ITC*'99 benchmark suite [9]. We considered two transformations: full TMR (*i.e.*, with triplicated voters after each memory cell) and DTR as described before. Each transformed circuit was synthesized for FPGA using *Synplify Pro* without any optimization (resource sharing, FSM optimization, etc.). We have chosen flash-based ProASIC3 FPGA family as a synthesis target. Its configuration memory is immune to soft-errors [7] and data memory is protected with one of the above transformations.

The circuits are sorted according to the number of core cells of the original circuit after synthesis. Figures 8 shows the results for the largest circuits and 9 shows the results for the smallest ones.

The DTR circuits require significantly less hardware for almost all circuits of the suite. Since the technique reuses the combinational part, hardware benefits are growing with the size of the combinational part. The constant hardware cost of the supporting mechanisms (control block, input/output buffers) becomes negligible when the size of the original circuit is large enough.

Figure 8 shows that the DTR circuits are 1.39 to 2.1 times larger than the original ones. For comparison, TMR circuits are 3.3 to 3.9 larger than the original ones. The largest hardware overhead for all circuit transformations has been observed for $b12$ circuit, a game controller with 121 memory

cells [9]. The TMR and DTR version of $b12$ are respectively 3.9 and 2.1 times larger than the original circuit.

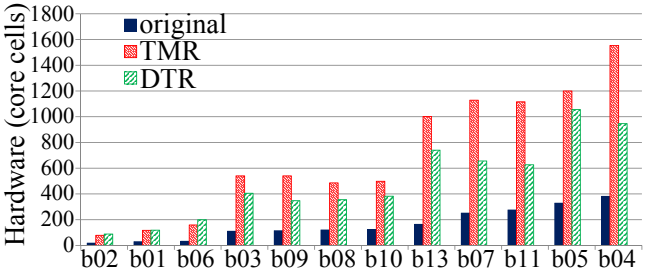

Figure 9: Circuit size after transformation (small circuits).

Figure 9 shows that, for the majority of the smallest circuits (< 100 memory cells), DTR still have less hardware overhead than TMR. But this benefit is negated for the tiny circuits $b01$, $b02$, and $b06$ (< 10 memory cells) due to the hardware overhead of the control block and input/output buffers. For such small circuits, TMR is clearly a better option.

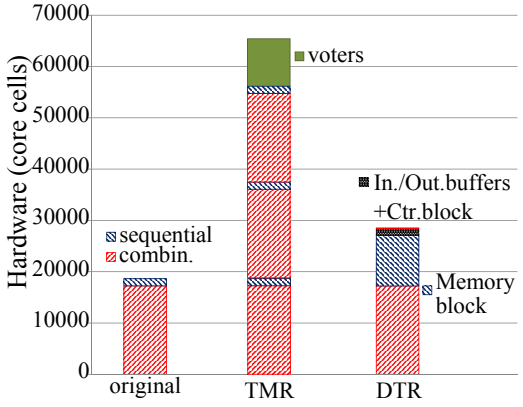

Figure 10: Transformed circuits profiling (for *b17*).

Figure 10 demonstrates why DTR transformation has significantly less hardware overhead compared to TMR. The synthesized circuit $b17$ (first bar) consists of a large combinational part (bottom part: 17240 core cells) and a small sequential part (top part: 1415 core cells). In the TMR version of $b17$ (second bar) the triplicated combinational part is dominant. The triplicated voters after each memory cell (for protection against SET) occupy 14.5% of the whole circuit. The DTR circuit (third bar) reuses the combinational part, so its size stays the same. The hardware cost of the DTR control block, input and output buffers is negligible (only 0.05% of all needed hardware resources). As a result, after DTR transformation, the circtuit occupies 153% of its original size (hardware overhead = 53%) whereas, after TMR, it takes 350% (harware overhead = 250%). The overhead of DTR is 4.7 times smaller than TMR for $b17$ and between 2.7 to 6.1 times smaller for the whole *ITC*'99 benchmark suite.

Although DTR has a significantly smaller hardware overhead than TMR it decreases the circuit throughput. Indeed, since the technique requires the input streams to be upsampled, the throughput of the transformed circuit is at least divided by two. Figure 11 shows the ratio of the transformed circuit throughput *w.r.t.* the corresponding original throughput for the ITC'99 benchmark suite (sorted left to right *w.r.t.* the size of the original circuit). Throughput is defined as the number of significant bits (*i.e.,* those not created by the upsampling) processed per time unit. Besides the upsampling, the DTR transformation influences by itself (as well as TMR) the circuit maximum frequency, which also changes the final throughput. In particular, the maximum synthesizable frequency after DTR transformation reaches ∼75% of the original frequency for small circuits (for TMR it is ∼77%) and ∼92% for large circuits (for TMR it is ∼93%).

TMR voters clearly slow down the circuit: the decrease of throughput varies from 3–10% for large circuits (*e.g.,* $b17$, $b20 - b22$) to 25–35% for small ones (*e.g.,* $b02$, $b06$, $b03$). In the best case, the throughput of DTR circuits can reach 50% of the original circuit due to the double upsampling of inputs. The control block and the multiplexers in memory blocks also introduce a small extra overhead. For large circuits, the throughput is 40–50% of the original, while for small circuits it drops to 30–40%.

5. RELATED WORK

The principles of time-redundancy, check-pointing and rollback have been implemented in various methods. However, the existing schemes are either ASIC-oriented requiring a strong control on the clock lines, or do not tolerate SETs.

An application-oriented use of time-redundancy can be found in fault-tolerant designs of arithmetic units [10, 11] and CPUs [12, 13]. In the former case, the regularity of an arithmetic operation is used to organize an optimized unit architecture with error-detection and error-masking characteristics. For ASICs, the hardware overhead (up to 72%) and the throughput overhead (up to 19%) depend on the number of bits of the operation. These techniques are limited to specific application domains. In the latter case, a CPU is reorganized so that it executes each instruction twice with a further comparison and a rollback upon an error detection in order to self-recover [12]. The hardware overhead for FPGAs is 50-60% with only 79% of fault coverage (the performance penalty is not indicated). Time-redundancy is used at a higher level in [13]: the thread level. Time overhead is stated to be 30% while fault-coverage is not given. In both cases, these techniques, dedicated to CPUs, require the knowledge of the macro-architecture and do not guaranty full error masking.

Some ASIC-oriented time-redundant techniques rely on latching-window masking when an SET glitch is not latched by memory cells, since it does not satisfy setup and hold time conditions. Nicolaidis *et al.* [14, 15] present such an approach with a hardware overhead of 20-50% and 97-100% error-detection efficiency. The performance overhead is comparable to the one of TMR. A similar technique has been presented in [16] for SRAM-based FPGAs. However, no indication is given about its throughput overhead and nor how to organize three independent clock lines shifted relatively to each other to guarantee SETs masking. The latching-window principle is commonly used in ASIC CPU pipelines [17–19]. A "shadow" latch with its own clock line is attached to each original memory cell to create an error detection mechanism through time properties tuning. Being developed to tolerate soft-errors to organize a safe voltage

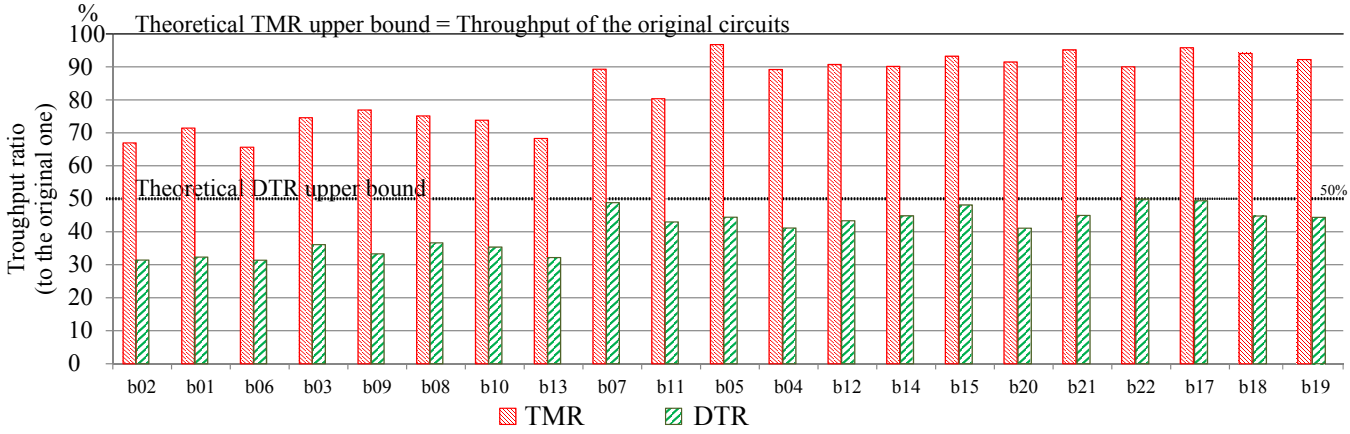

Figure 11: Throughput ratio of TMR, and DTR transformed circuits.

scaling, these techniques have a performance penalty as little as 3% providing near 100% fault-masking. However, all the mentioned restrictions (precise time properties tunings, additional clock lines, pipelined architecture) prevent the use of these approaches for FPGAs, where special circuitries to implement these techniques are not normally available.

General check-pointing/rollback techniques have been proposed in [20, 21]. As in our technique, it is implemented using a micro-architectural transformation. However, the resulting circuit is tolerant to SEUs but not SETs, regardless of the error detection mechanism used. Indeed, an SET may corrupt both a cell (*i.e.,* the current state) and its copy (*i.e.,* its check-point). As a result, when an error is detected, the rollback may return the circuit to an incorrect state. Our implementation excludes such behavior, because an error is always detected before a potentially corrupted image (contained in r cells, see Fig. 4) propagates to the checkpoints (represented by r' cells).

A Synplicity Inc. patent [22] shows how time-redundancy can be implemented through a time-multiplexed combinatorial circuit. This technique is close to ours in that it replaces all memory cells by a circuit similar to our memory block. However, it does not mask all possible SETs and uses triple (or higher) time redundancy. Using micro-level checkpointing/rollback (as well as a speedup mode and IO buffers), we can implement stronger fault-tolerance guarantees using double-time redundancy only.

6. CONCLUSION

We proposed a novel and general logic-level circuit transformation to automatically introduce time-redundancy for fault-tolerance in digital circuits. Our DTR technique guarantees that the transformed circuits are tolerant towards any SET (*a fortiori* SEU) with no influence on output streams, provided that errors occur less frequently than one every 10 clock cycles. The DTR transformation is formally provable which is important for safety-critical systems. Our approach is technologically independent, does not require any specific hardware support, and is suitable for stream processing.

The technique is based on the time-redundancy principle where a combinational circuit is time-multiplexed. Our approach associates in a new way several techniques: double-time redundancy, error detection, micro-level checkpointing/rollback, a speedup mode, and input/output buffers. The transformations substitute the original memory cells with a small circuit (a memory block), which introduces a time-redundant masking mechanism in an automatic manner.

In order to verify the fault-tolerance of the transformed circuit, we exhaustively checked all possible occurrences of SETs. We have applied the proposed transformations to the $ITC'99$ benchmark suite. Experimental results for flash-based FPGA synthesis show that the hardware overhead of DTR is 2.7 to 6.1 times smaller compared to TMR. It makes the overall size of DTR circuits 1.9 to 2.5 times smaller than TMR for circuits with more than 100 memory cells. DTR is less beneficial for small circuits (less than 100 memory cells and small combinational logic).

Such significant hardware reduction has been achieved with a trade-off on the transformed circuit's throughput. The throughput of a DTR circuit is 50–55% of the corresponding TMR circuit throughput. However, other existing time-redundant error-correcting techniques implementable in COTS FPGAs have higher throughput loss. However, the proposed technique is harder to apply selectively to sub-circuit as it can be done with TMR. The input/output streams up-sampling may lead to adjustments of surrounding circuits.

Our transformation is an alternative to TMR for any logic intensive circuits where hardware overhead takes precedence over throughput. As in software, hardware time redundancy is only suited to applications that do not require high 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 DTR technique.

An easy extension to DTR allows the surrounding circuit to switch off time redundancy (as the control block does in the recovery phase). This feature permits to dynamically and temporarily give up fault-tolerance and speed up the circuit. The motivations for such changes can be based on the observed change in radiation environment or the processing of (non)critical data.

In addition, DTR introduces new pipelined cells which may allow optimizations using retiming. In particular, mem-

ory cells r and d might be moved into the combinatorial circuit to break its critical path.

In this paper, we did not consider SETs on the clock line. DTR could be made tolerant to this kind of faults by using two independent and synchronous clocks. They would be used to prevent the simultaneous corruption of particular memory cells and guarantee error detection and proper recovery. For instance, memory cells (d, d') should use two different clocks to avoid their simultaneous corruption that would prohibit error-detection.

Finally, even if we have manually checked all possible occurrences of SETs and performed fault-insertions on a few transformed circuits with the *ModelSim* simulator, a mechanically checked proof is necessary for a complete assurance. We have started the formal certification of such transformations using the Coq proof assistant [23]. We describe program transformations on a simple gate-level hardware description language and the fault-model is described in the operational semantics of the language. The main theorem states that, for any circuit, for any input stream and for any SET allowed by the fault-model, its transformed version produces a correct output. We have already used this approach to prove the correctness of transformations implementing TMR and triple time redundancy. At the time of writing, the proof of the DTR transformation is in progress.

7. 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. D. Bradley and E. Normand, "Single event upsets in implantable cardioverter defibrillators," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2929–2940, 1998.

[3] 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 *Int. Conference on Dependable Systems and Networks*, 2002, pp. 389–398.

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

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

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

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

[8] 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*, vol. 58, no. 6, pp. 2804–2806, 2011.

[9] F. Corno, M. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *Design Test of Computers, IEEE*, vol. 17, no. 3, pp. 44–53, 2000.

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

[11] W. Townsend, J. Abraham, and E. Swartzlander, "Quadruple time redundancy adders [error correcting adder]," in *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, Nov 2003, pp. 250–256.

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

[13] E. Rotenberg, "Ar-smt: a microarchitectural approach to fault tolerance in microprocessors," in *Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing.*, June 1999, pp. 84–91.

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

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

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

[17] D. Ernst *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Int. Symp. on Microarchitecture, MICRO-36.*, Dec 2003, pp. 7–18.

[18] N. Avirneni, V. Subramanian, and A. Somani, "Low overhead soft error mitigation techniques for high-performance and aggressive systems," in *Int. Conf. on Dependable Systems Networks*, June 2009, pp. 185–194.

[19] G. Sohi, M. Franklin, and K. Saluja, "A study of time-redundant fault tolerance techniques for high-performance pipelined computers," in *FTCS-19*, June 1989, pp. 436–443.

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

[21] D. Koch, C. Haubelt, and J. Teich, "Efficient hardware checkpointing: Concepts, overhead analysis, and implementation," ser. FPGA '07, 2007, pp. 188–196.

[22] K. S. McElvain, "Circuits with modular redundancy and methods and apparatuses for their automated synthesis," *Synplicity, Inc. Patent No.US007200822B1*, April 2007.

[23] Coq development team. The coq proof assistant, software and documentation available at http://coq.inria.fr/, 1989-2014.