SEU-Aware Resource Binding for Modular Redundancy Based Designs on FPGAs

Shahin Golshan
Computer Sciences Department
University of California, Irvine
golshans@uci.edu

Eli Bozorgzadeh
Computer Sciences Department
University of California, Irvine
eli@ics.uci.edu

Abstract— **Although Triple Modular Redundancy (TMR) has been widely used to mitigate single event upsets (SEUs) in SRAM-based FPGAs, SEU-caused bridging faults between the TMR modules do not guarantee correctness of TMR design under SEU. In this paper, we present a novel approximation algorithm for resource binding on scheduled datapaths at the presence of TMR, which aims at containment of each SEU within a single replica of tripled operations. The key challenges are to avoid resource sharing between modular redundant operations and also to reduce the possibility of TMR masking breaches in resource allocation. We introduce the notion of *vulnerability gap* during resource sharing to potentially reduce the effort for white space allocation at the physical design stage in order to avoid bridging faults between TMR resources. The experimental results show that our proposed resource binding algorithm, followed by floorplanner, reduces the potential of TMR breaches by 20%, on average.**

Keywords: Triple modular redundancy; single event upset; high level design; FPGA

I. INTRODUCTION

SRAM-based FPGA devices are becoming more sensitive to soft errors due to shrinking feature size, high density, and lower operating voltages [1-4]. Functionality and connectivity of implemented circuits on SRAM-based FPGAs can be damaged if a SEU caused by soft errors flips a configuration bit. SRAM-based FPGAs are rather more vulnerable to SEUs than their ASIC counterparts [1-4]. This paper focuses on SEU mitigation in high level synthesis for SRAM-based FPGAs.

Several fault tolerance methods have been proposed in the past to mitigate the effects of SEUs in the configuration bits. Some techniques try to restore the correct value of such flips once an error has been detected [9]. Others try to filter out the propagation of faults to the outputs of the circuit using modular redundancy methods [12-16]. *Triple modular redundancy* (TMR) has been widely used to mask out faults provided that multiple errors can not emerge at the same time instant. The basic concept of TMR scheme is to design three copies of the same element and to put a majority voter on the output of the replicated elements.

The capability of TMR for FPGA designs in tolerating SEUs has been studied in [2-4]. It has been observed that TMR schemes are able to only partially mitigate the effects of SEUs in routing configuration bits, which comprises around 90% of all configuration bits. In fact, 10% of SEUs that affect configuration bits of routing resources produce multiple errors that TMR is not able to filter out [2-4]. Such failing cases indicate that TMR-based techniques by themselves are not sufficient to guarantee complete reliability. It is difficult to contain errors within a single replica of TMR at physical design stage, especially when the replicas are placed too close to each other, where SEU-caused bridging faults between the TMR modules do not guarantee correctness of the TMR design under SEU [2-4].

While several related works focus on analysis and removal of such TMR breaches in FPGAs in physical design stages [2, 3, 4, 15, 16], we extend the concept of containment of SEUs within a single TMR replica to high level synthesis stage. We also propose a SEU aware layout design stage which is coupled with high level synthesis in order to reduce the potential bridging faults in later stages.

Enforcing complete reliability in TMR schemes in high level design synthesis faces key challenges of how to avoid resource sharing between modular redundant operations and also how to reduce the possibility of TMR masking breaches in resource allocation. To the best of our knowledge, our work is the first high level synthesis technique that addresses resource binding to contain SEUs within a single replica of TMR operations. Our algorithm primarily minimizes the number of resources allocated to operations under the condition that no two replicas of the same operation can share a resource and secondly, it reduces the TMR masking breaches. We prove that our approximation uses only up to two more resources compared to the optimal solution (regardless of being partial or full TMR). We propose the notion of vulnerability gap which guides the floorplanner to dedicate white space in the layout to avoid bridging fault. We also show that by combining our resource binding algorithm with FPGA floorplanners, we can reduce the potential of TMR breaches by 20%, on average.

II. RELATED WORK

Several SEU mitigation techniques have focused on reconfiguring the faulty bits once a bit flip is observed [9], which try to restore the proper value of such bits as soon as possible. Other techniques try to lower the impact of such bit flips by lowering the susceptibility of figuration bits [10, 11], in which the physical design stage reduces the total number of susceptible configuration bits.

The analysis and application of modular redundancy hardening at different phases have been studied extensively in

978-3-9810801-5-5/DATE09 © 2009 EDAA

the last decade [12-16]. High-level synthesis has been the focus of [12, 13]. In [12], the authors proposed a behavioral synthesis tool to schedule tasks and insert voters into different regions of a data flow graph (DFG) based on their degree of fault observability while lowering the area overhead. In [13], the authors introduced new TMR task scheduling/binding strategies which target correlated as well as independent faults using markov models.

Unlike high-level design synthesis, modular redundancy techniques in placement and routing phases of FPGA designs often focus on small granularities (i.e. control logic blocks) [14, 15, 16]. Partial TMR in FPGAs has been investigated in [14], which applies TMR selectively depending on their sensitivities against soft errors and the persistence of such errors once emerged. The closest related work to this paper is [16], where a new reliability-oriented place-and-route algorithm for SRAM-based FPGAs has been introduced. The objective is to force the SEUs to be contained within single copy of TMR replicas at CLB level so that no SEU-caused bridging fault could affect more than one TMR replica. Our technique on the other hand moves the concept of containment of SEU-caused fault to a single replica of TMR to above RTL. Reducing the potentials of TMR breaches in high level synthesis lightens the burden of removing TMR breaches in placement and routing stages. By sharing resources along with reducing TMR breach potentials, we can save area and power in comparison to assigning distinct resources to TMR replicas resulting in 3x resources.

III. CHALLENGES OF MODULAR REDUNDANCY BASED RESOURCE BINDING IN FPGA DESIGN

In this section, we explain the key challenges undermining the integrity of TMR schemes in FPGA high level design.

A. *Modular Redundancy Conflicts*

Configuration of a data path resource can get affected if soft error leads to a bit flip in configuration bit stream. We need to make sure that only one copy of the *modular redundant* (MR) operation is affected. Hence MR operations cannot be time-shared. If the SEU incurred in a configuration bit of a resource is not resolved in time, it will affect not only the current operation executing on the resource, but also the operations scheduled to be executed on the same resource afterwards (SEUs in configuration bits are not transient). As a result, in resource sharing stage, only operations that do not belong to modular redundancy set of the same operation are eligible to be considered for resource sharing. We refer to this type of conflicts among operations belonging to same MR as *modular redundancy conflicts*.

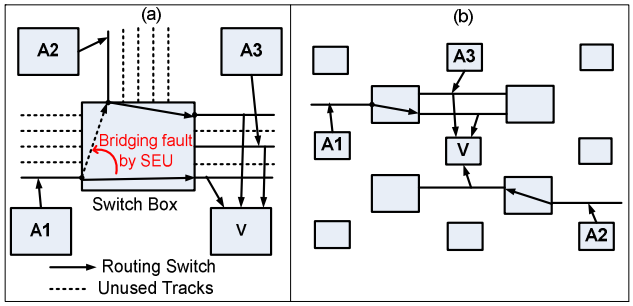

Figure 1. SEU impact when TMR replicas are placed (a) closer to one another and (b) farther to one another.

B. *Vulnerability Gap Conflicts*

An SEU in FPGA routing architecture may connect two nets of different resources, which execute operations that are redundant copies of the same operation (A1 and A2 in Figure 1.a.). Bridging faults caused by SEU corrupt the functionality of both resources and result in unreliable outputs. Hence, physical design tools need to confine the error to a single resource. If the nets of the two resources do not share any adjacent routing resource, there is no room for such faults. Therefore, the routing tool must route the nets in such a way that it does not let any two nets of two resources of the same redundancy set go through the adjacent routing resources. The router is unable to do so unless sufficient white space is provided for the router to maneuver the nets of the modules of the same redundancy set [16]. If white space is accommodated between resources holding two operations of the same MR set, redundancy-aware physical design tools can handle the restrictions using the extra space provided [16]. As shown in Figure 1.a, when modular redundant replicas A1 and A2 are placed close to each other, bridging faults might appear due to SEU in routing architecture. As we move A2 to a farther location, the bridging fault can be easily avoided in the routing stage (Figure 1.b).

We define *vulnerability gap* (VG) between two resources based on the operations each resource is bound to. If two resources do not hold operations from a common MR set, VG is zero. Otherwise there is a VG between the two resources. The VGs caused by modular redundancy must be respected by layout design tools. Respecting VGs adds additional burdens to physical design tools to minimize wirelength and critical path delay.

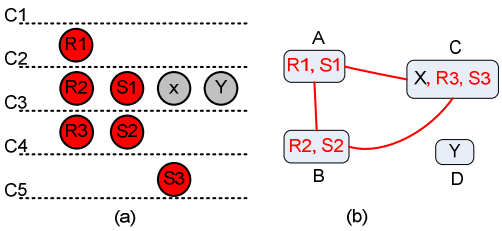

Figure 2. (a) Scheduled TMR-based data flow graph
(b) Optimal resource binding in terms of vulnerability gap

Once operations are scheduled and bound to resources, VG constraints should be enforced between such resources in placement so that sufficient white space is allocated. This effort is in conflict with conventional physical design objectives which tend to push the resources closer together (i.e. wirelength, critical path delay and area minimization objectives). However, if VG conflicts between resources are reduced in resource binding stage, physical design tools have an easier time to resolve VG violations and to focus on optimizing conventional objectives.

Figure 2 provides a clarifying example of the impact of resource binding in VG reduction. The MR operations are

depicted in red. The sets {R1,R2,R3} and {S1,S2,S3} are the two MR sets in this example. Two different binding solutions bind four resources {A,B,C,D} to scheduled operations in Figure 2. The first resource binding is as follows: A←{R1,S}, B←{R2,S2}, C←{X,R3,S3}, D←{Y}. In this binding, there is a need for VGs between three resource pairs: (A,B), (A,C) and (B,C). The VG constraints of this resource binding are illustrated in Figure 2 b. The second resource binding is as follows: A←{R1,S1}, B←{R2,S2}, C←{X,R3} and D←{Y,S3}. In this binding, there are vulnerability gap constraints between five resource pairs: (A,B), (A,C), (A,D), (B,C) and (B,D). In the second binding, more vulnerability gap constraints must be respected during place and route and the place and route tools face more challenges in avoiding routing adjacency between the resources within VG. For example, in the second resource binding, bridging faults between the nets of resources D and A can damage the functionality of both resources and therefore the TMR scheme might fail. The same argument is true for the nets of resources D and B. However, in the first binding, SEU-caused bridging faults between nets of resources A and D damage the functionality of two resources belonging to two different redundancy sets. In other words, only a single replica of each redundancy set is damaged and the TMR scheme can filter out such faults.

In this paper, we extend our proposed MR-based resource binding algorithm to help enhance the confinement of SEUs to a single replica at physical design stage. In our proposed resource binding, the objective is to minimize the number of vulnerability gap constraints while resolving timing and MR conflicts.

IV. SEU-AWARE RESOURCE BINDING

Throughout this section it is assumed that all the operations are scheduled before resource binding. Hence each operation O_i has a start time and an end time (S_i, E_i) in terms of cycles. In order to elaborate the proposed solution we first define the terms used throughout this paper:

Redundancy Group: A set of operations that cannot share a resource since they belong to the same MR set. For example, in Figure 2, the sets $\{R_1, R_2, R_3\}$ and $\{S_1, S_2, S_3\}$ comprise two redundancy groups.

Redundancy Degree (D): Number of operations in each redundancy group (in TMR design, this degree is 3).

Redundancy ratio (for partial modular redundant designs): The ratio of operations duplicated to form a redundancy group to the total number of operations in the design. In Figure 2.a, the redundancy ratio is 2/4

The problem of resource binding under timing and modular redundancy conflicts can formally be stated as: Given a set of timing intervals corresponding to scheduled operations and a set of MR conflicts between the operations, assign the operations to the minimum number of resources such that no two operations assigned to the same resource have timing or modular redundancy conflicts and minimize the number of VG constraints between resources.

If resolving timing conflicts is the only concern of resource-binding, the solution to the problem of resource-binding is well known. Since the conflict graph modeling the conflicts between operations is an *Interval* graph [5], and polynomial-time algorithms such as Left-edge algorithm [6], can be employed to solve resource binding problem optimally. However, when we include MR conflicts in the conflict graph, the conflict graph no longer belongs to the class of interval graphs. In Figure 3.a, the operations X1, X2 and X3 are three operations in the same redundancy group. In Figure 3.b, the corresponding conflict graph contains a cordless cycle of length four that makes the graph non-interval [5].

The knowledge on start time and end time of the operations implies a chronological order in which the operations are executed on each resource. We have developed an approximation algorithm based on LEA that greedily binds operations to resources while avoiding redundancy and timing conflicts, where the objectives are primarily minimization of the number of resources used and secondarily, reduction of the

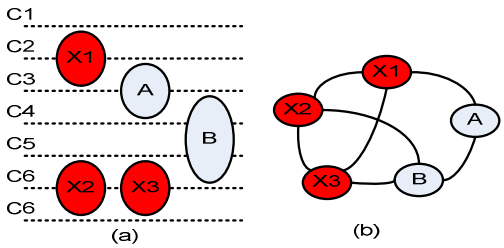

Figure 3. (a) Scheduled operation, where X1, X2 and X3 operations are the copies of the same redundancy group.
(b) Conflict graph containing timing and MR conflicts

VG constraints among resources. It starts from a single resource. Then it picks the operations according to the sorted list of operations one at a time. It ranks all the available resources that do not conflict with the current operation in terms of redundancy and timing conflicts, based on the number of new VGs the binding generated. The resource generating minimal VGs is picked. If all current resources conflict with the operation, a new resource is bound to the operation.

At the worst case, the solution provided by our algorithm uses $(D - 1)$ more resources than the number of resources used in the optimal solution. If we are using TMR (or partial TMR) scheme, our approximation algorithm generates a solution which in the worst case, uses 2 extra resources. The pseudo-code of our algorithm is illustrated in Figure 4.

Theorem 1. Given that the optimum solution to resource binding problem under timing and MR constraints use M resources, the solution obtained from the algorithm *Allocate-Resource* uses at the worst case $M + D - 1$ resources.

Proof. Due to simplicity, we only provide the proof for the case where D is 3. That is, each operation belonging to a redundancy group conflicts with two other operations.

Let us assume that the number of resources used in *Allocate-Resource* algorithm exceeds the bound of $M + 2$ and O_i is the first operation to be assigned to the $(M + 3)^{th}$ resource in the course of execution of the algorithm. Since the degree of redundancy is 3, each operation belonging to a redundancy

group conflict with two other operations in the design. If O_i does not belong to any redundancy group, then the only reason that the $(M + 3)^{th}$ resource has been assigned to O_i is that each of resources 1 through $(M + 2)$ contains operations that conflict with O_i in terms of their timing interval. All the operations currently conflicting with O_i must have start times less than or equal to S_i, and must have finish times after S_i. These operations not only conflict with O_i, they all conflict with each other. Hence, it is impossible for us to bind all the operations to M resources.

Procedure Allocate-Resource

Inputs:
-A set of operations, *S*, such that each operation has a start time and a finish time
-A set of redundancy groups, where each operation may belong to only one redundancy group.

Output: A conflict-free assignment of operations to resources, with minimal number of resources (*m*).

$m \leftarrow 0$
While $S \neq \Phi$
  Remove from *S* the operation O_i with the smallest start time
  $min\_cost \leftarrow \infty$
  $min\_index \leftarrow 0$
  For $j \leftarrow 1$ to m do
    if $cost(R_j, O_i) < min\_cost$ then
      $min\_cost \leftarrow cost(R_j, O_i)$
      $miin\_index \leftarrow j$
  if $min\_cost < \infty$
    assign R_{min_index} to O_i
    $update\_vulnarabilty\_gap(R_{min\_index}, O_i)$
  else
    $m \leftarrow m + 1$
    assign R_m to O
    $update\_vulnarabilty\_gap(R_m, O_i)$

Function Cost

Inputs: resource R_j, operation O_i
Outputs: the cost of assigning resource R_j to operation O_i
$cost \leftarrow 0$
If there is a conflict (timing/redundancy) conflict between R_j and O_i
  $cost \leftarrow \infty$
  exit
let *RG* be the redundancy group to which O_i belongs
for all operations O_x in *RG* do
  let R_x be the resource assigned to O_x
    if $vulnarabilty\_gap(R_j, R_x) = 0$ then
      $cost \leftarrow cost + 1$

Procedure Update_Vulnarabily_Gap

Inputs: resource R_j, operation O_i
let *RG* be the redundancy group to which O_i belongs
for all operations O_x in *RG* do
  let R_x be the resource assigned to O_x
    if $vulnarabilty\_gap(R_j, R_x) = 0$ then
      $vulnarabilty\_gap(R_j, R_x) \leftarrow 1$

Figure 4. Outline of resource binding algorithm under timing/redundancy constraints

The same argument can be used for the cases where O_i belongs to a redundancy group and there are 2 resources (or 1 resource) that conflict with O_i in terms of redundancy constraints. The reason O_i uses the $(M + 3)^{th}$ resource is that there are M resources which conflict with O_i in terms of timing constraints and 2 resources which conflict with O_i in terms of modular redundancy constraints. Using the same reasoning, we can show that we need $(M + 1)$ resources to resolve timing conflicts among such operations and this contradicts the assumption that the optimal resource binding solution uses M resources. □

Corollary. For TMR designs (D=3), the proposed algorithm always uses up to two more resources compared to the optimum resource binding solution independent of redundancy ratio.

In order to gain insight about the behavior of our algorithm, we apply our algorithm on Figure 2. According to their start times, the sorted list of the operations in Figure 2 is $\{R_1,R_2,S_1,X,Y,R_3,S_2,S_3\}$. When the algorithm reaches R_3, the unique assignment of resources to the preceding operations is: $A \leftarrow \{R_1, S_1\}$, $B \leftarrow \{R_2\}$, $C \leftarrow \{X\}$ and $D \leftarrow \{Y\}$, and the only VG is between resources A and B. The only available resources for R_3 are C and D. R_3 is in conflict with resources A and B and hence, the cost of picking either C or D is 2. If C is assigned to R_3 : $C \leftarrow \{X, R_3\}$, VGs are between A and B, between A and C, and between B and C. As the algorithm reaches S_2, the available resources will be B and D. The cost of picking B is zero, since picking B does not add a new VG between any two resources (since all VGs have already been considered between A, B and C, generated by redundancy group $\{R_1,R_2,R_3\}$). However, the cost of picking D is 1 since picking D adds a new VG between D and A. Therefore B is picked: $B \leftarrow \{R_2, S_2\}$. Similarly, the algorithm picks C for S_3. Adding S_3 to C does not add any new VG, whereas adding S_3 to D adds two new VGs between D, A and B because of redundancy group $\{S_1,S_2,S_3\}$. The final assignment of resources to operations is depicted in Figure 2.b.

Theorem 2. The worst case execution time of our resource binding algorithm is $O(D.n^2)$.

V. VULNERABILITY-GAP CONSIDERATION IN FLOORPLANNING

Once the circuit blocks (resource and block are used interchangeably in this context) have been arranged according to the high-level design specification, we need to specify the layout of the design on a chip. This procedure is carried out in the floorplanning stage.

The primary objective of floorplanners is to reduce the total area/wirelength of a design. Many floorplanners use simulated annealing framework It has been shown in [7, 8], sequence pair representation is an efficient and suitable representation that allows effective traversals of the solution space through simple moves/cost calculations using simulated annealing engines.

A sequence pair representation consists of two permutations of N blocks. The two permutations capture the geometric relations between each pair of blocks. In other words, every two blocks constrain each other in either horizontal or vertical direction [7]. The following conditions define the horizontal/vertical constraints between the blocks:

X:(<...,*a*,...,*b*,...>,<...,*a*,...,*b*,...>)→ *a* is to the left of *b* ,

Y: (<...,*a*,...,*b*,...>, <...,*b*,...,*a*,...>) → *a* is above *b*.

In order to realize the actual layout a sequence pair represents in the minimum area possible, a simple algorithm has been proposed in [7], which computes the X-locations and Y-locations of the blocks independently using the horizontal (H) and vertical (V) constraint graphs obtained from the sequence pair. Every node represents a block in the *V*(*H*) graph. For every pair of blocks *a* and *b*, there is a directed edge *a*→*b* in the *H*(*V*) graph if *a* is to left(above) *b*. The

weights in H graph horizontal sizes of blocks and the weights in the V graph represent vertical size of the blocks. Applying the longest path algorithm on V and H graphs, we obtain the dimensions of the smallest chip area containing all the blocks.

The constraints implied by a sequence pair guarantee that no two blocks overlap. But these constraints do not guarantee that certain blocks are not too close to each other. As mentioned earlier, VG constraints imply that there should be some extra distance between certain blocks that contain operations of the same redundancy group or otherwise, a soft error might not be contained within a single copy of the redundant operation.

Function EnumerateViolations
Inputs:
Sequence pair:(X, Y)
Block locations, widths and heights: *xloc*, *yloc, width, height*
List of vulnerability-gap conflicting blocks: *conflict*(B_i,B_j)
Min. required distance between conflicting blocks: *v_dist*
Output:
the number of vulnerability-gap violations in the layout: *v_count*

```
v_count ← 0
for i ← 1 to sizeof(X) do
  for j ← 1 to i-1do
    if conflict(Bi,Bj) is true
      if Bi procedes Bj in Y then
        if (xloc(Bi) < xloc(Bj) + width(Bj) + v_dist) then
            v_count ← v_count +1
      else
        if (yloc(Bj) < yloc(Bi) + height(Bj) + v_dist) then
            v_count ← v_count +1
```

Figure 5. Outline of *EnumerateViolations*

In order to reduce the number of vulnerability gap violations in the design, we have added a new cost function to simulated annealing engine along with area and wirelength cost functions. This cost function calculates the number of redundancy gap violations existing in the layout realized by the current sequence pair. A violation in vulnerability gap constraint between two blocks is realized if they are closer than vulnerability gap distance *v_dist*. As outlined in Figure 5, the function *EnumerateViolations* counts the total number of VG violations through counting the total number of times two conflicting blocks are closer than *v_dist* threshold.

VI. EXPERIMETNS

In order to evaluate our resource binding algorithm and violation-aware floorplanner, we have set up the experimental flow as illustrated in Figure 6. The benchmarks used in our experiments are DSP and multimedia applications, widely used in high-level synthesis community [17]. The operations used in the data flow graphs are adders, multipliers, dividers, shifters, majority voters and logic operations. Each operation was synthesized using Xilinx Core Generator version 9.2 for Virtex IV target device for 8-bit data paths.

In each experiment, based on TMR ratio, we randomly selected some operations and tripled them. We have implemented list scheduling algorithm [6], which minimizes the number of resources in the design under minimum latency. In order to evaluate the performance of our resource binding algorithm, we have compared our algorithm with another variant of our algorithm as the reference case, which does not consider modular redundancy cost reduction (vulnerability gap in Figure 4) Both resource binding algorithms result in the same number of resources.

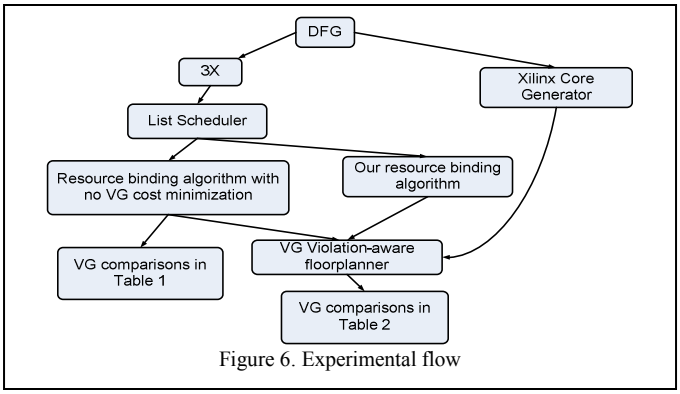

Figure 6. Experimental flow

We have implemented our vulnerability-gap-aware floorplanner on top of UMpack [18], which is the successor of the popular academic floorplanner tool, Parquet. We set the distance between conflicting blocks *(v_dist)* to be five slices. The tool is set to reduce the area, wire length and the vulnerability gap violations.

In Table 1, we report the total number of VG constraints generated by the two binding algorithms. The test benches are depicted in Table 1 based on their sizes from left to right. The experiments are carried out for 10%, 20%, and 50% TMR ratios. For 10%, 20% and 50% TMR ratios, improvements are 26%, 23%, and 12%, respectively. The maximum reduction reached by our algorithm is 50% for DCT_FAST test bench when the TMR ratio is 10%.

As the TMR ratio goes higher more resources get involved in some redundancy group, and therefore the total number of conflicts increases. For example, a TMR ratio of 50% implies that half of the operations in the DFG are tripled. Note that our resource binding algorithm primarily resolves the modular redundancy conflicts and timing conflicts in assigning resources to operations and then tries to reduce the VG constraints among resources. When the TMR ratio is low, since there is more room for the binding algorithm to rank the resources (according to the cost function of Figure 4), there is an impressive reduction rate.

Another important factor in reducing the VGs between the resources is the number of available resources for the algorithm at each step of execution. For DFGs which are highly parallel and the number of operations are high, VG reduction is more drastic. Therefore, as we move to the right in the table, with few exceptions the improvements go higher.

In the next set of experiments, we use the vulnerability-gap-aware floorplanner to evaluate the actual number of VG violations in the floorplan (Table 2). The average improvement in violation gap reduction is 19%, 21%, and 12% for 10%, 20%, and 50% TMR ratios, respectively. The maximum amount of reduction is 44% in the case of IDCT when the TMR ratio is 10%. In Table 2, the actual number of VG violations is lesser than the number of VG constraints among resources in Table 1. The main reason is that the VG conflicts are between any two resources with modular redundancy constraint whereas VG violations emerge only

when the two resources are closer than a certain distance (*v_dist*). In addition to VG reduction, the simulated annealing tool also tries to minimize the wirelength and the whitespace among the resources. In a few cases where there is no improvement in terms of vulnerability gap violations, the other layout parameters are improved. For example in 10% TMR ratio, FIR2 and FDB, show 17% and 5% improvement in the wirelength and 3% and 15% improvement in white space percentage.

We also explored the floorplanner results in terms of white space percentage and wire length values. On average, there is a slight increase in the white space percentage (<2%), while we improve the wire length of the designs 5%, on average. In the best cases, we improved the wirelength by 20% and the white space percentage by 25%.

VII. CONCLUSIONS

In this paper, we presented a novel approximation resource binding algorithm that primarily minimizes the number of resources allocated to operations under the condition that no two replicas of the same operation can share a resource and secondly, it reduces the TMR masking breaches. when coupled with vulnerability-gap-aware floorplanner, our resource binding algorithm reduces the vulnerability gap violations by 20% on average and in the best case by 44%.

REFERENCES

[1] P. Graham et al., "Consequences and categories of SRAM FPGA configuration SEUs", MAPLD'03

[2] P. Bernardi et al., "On the evaluation of SEU sensitiveness in SRAM-based FPGAs", IOLTS'04

[3] G. Swift et al., "single-event upset susceptibility testing of the Xilinx Virtex II FPGA", MAPLD'02

[4] N. Rollins et al., "evaluating TMR techniques in the presence of single event upsets", MAPLD'03

[5] M. C. Golumbic, **"**Algorithmic graph theory and perfect graphs" Algorithmic graph Theory and perfect graphs" , 2nd ed., Elsevier, 2004

[6] G. De Micheli, "Synthesis and optimization of digital circuits", McGraw-Hill, 1994

[7] H. Murata et al., "VLSI module placement based on rectangle-packing by the sequence pair", *IEE Trans. on CAD*, vol. 15(12), pp. 1518-1524, 1996

[8] S. N. Adya et al., "Fixed-outline floorpalnning through better local search", IEEE, Sept. 2001

[9] C.Carmichael et al., "Correcting Single-Event-Upset through Virtex partial reconfiguration", Xilinx application notes, XAPP216, 2000

[10] Zarandi, et al. "SEU-mitigation placement and routing algorithm and their impact in SRAM-based FPGAs", ISQED'07

[11] S. Golshan et al., "Single-Event-Upset (SEU) awareness in FPGA routing", DAC'07

[12] R. Karri, et al., "time-constrained scheduling during high-level synthesis of fault-secure VLSI digital signal processors", *IEE Trans. On Reliability*, Sep. 1996

[13] S. Woo et al., "Task-scheduling strategies for reliable TMR controllers using task grouping and assignment", *IEE Trans. On Reliability*, Dec. 2000

[14] B. Pratt et al., "improving FPGA design robustness with partial TMR", *Reliability Physics Symposium Proceedings,* 2006

[15] F. Lima, et al., "on the optimal design of triple modular redundancy logic for SRAM-based FPGAs", DATE'05

[16] L. Sterpone, et al., "a new reliability-oriented place and route algorithm for SRAM-based FPGAs", *IEEE Trans. On Computers,* vol. 55, NO. 6, June 2006

[17] Express group at UCSB: http://express.ece.ucsb.edu/benchmark/

[18] http://www.openedatools.org/projects/umpack/

Table 1. Vulnerably gap conflicts among resources

TMR	Tool	Hal	Horner	arf	motion	ewf	fir1	fir2	fdb	smooth down samule	collapse	cos1	cos2	bmp	inter-polate	matmul	idct	dct_fast	dct slow	smooth triangle	invert_ matrix
10%	new	46	50	53	46	53	46	46	88	43	53	86	87	135	223	156	239	117	132	199	343
	ref.	53	53	68	55	76	61	50	109	59	101	89	94	176	259	214	336	258	266	349	570
	%	**13**	**6**	**22**	**16**	**30**	**25**	**8**	**19**	**27**	**48**	**3**	**7**	**23**	**14**	**27**	**29**	**55**	**50**	**43**	**40**
20%	new	46	66	92	129	95	46	53	118	74	162	154	134	298	294	280	409	358	328	496	641
	ref.	53	71	101	155	107	61	71	150	104	225	178	173	344	417	375	549	486	515	687	1089
	%	**13**	**7**	**9**	**17**	**11**	**25**	**25**	**21**	**29**	**28**	**13**	**23**	**13**	**29**	**25**	**26**	**26**	**36**	**28**	**41**
50%	new	93	88	195	286	159	132	131	363	236	394	361	360	809	761	696	901	776	906	1427	2070
	ref.	102	88	226	311	170	136	147	396	253	452	433	391	839	916	823	1083	953	1086	1645	2520
	%	**9**	**0**	**14**	**8**	**6**	**3**	**11**	**8**	**7**	**13**	**17**	**8**	**4**	**17**	**15**	**17**	**18**	**17**	**13**	**18**

Table 2. Vulnerably gap violations among resources reported by floorplanner

TMR	Tool	Hal	Horner	arf	motion	ewf	fir1	fir2	fdb	smooth down samule	collapse	cos1	cos2	bmp	inter-polate	matmul	idct	dct_fast	dct slow	smooth triangle	invert_ matrix
10%	new	21	31	18	27	22	32	29	36	24	27	34	31	62	54	45	58	47	41	60	62
	ref.	34	33	29	33	31	32	28	35	29	39	45	32	79	76	51	103	48	70	69	87
	%	**38**	**6**	**38**	**18**	**29**	**0**	**-4**	**-3**	**17**	**31**	**24**	**3**	**22**	**29**	**12**	**44**	**2**	**41**	**13**	**29**
20%	new	21	46	42	52	36	38	35	35	45	52	49	42	104	70	79	87	80	77	95	79
	ref.	34	53	63	61	53	41	39	54	58	68	49	49	140	105	100	98	91	107	105	126
	%	**38**	**13**	**33**	**15**	**32**	**8**	**10**	**35**	**22**	**24**	**0**	**14**	**26**	**33**	**21**	**11**	**12**	**28**	**10**	**37**
50%	new	44	46	63	97	63	49	52	72	74	110	93	90	204	134	105	147	128	105	169	185
	ref.	49	68	70	100	64	49	63	93	79	112	106	96	236	174	137	159	137	137	181	219
	%	**10**	**32**	**10**	**3**	**2**	**0**	**17**	**23**	**6**	**2**	**12**	**6**	**14**	**23**	**23**	**8**	**7**	**23**	**7**	**16**