2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

Hypergraph-Cover Diversity for Maximally-Resilient Reconfigurable Systems

Ahmad Alzahrani and Ronald F. DeMara
Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816-2362
Email:{azahrani@knights, demara@mail}.ucf.edu, http://cal.ucf.edu

***Abstract*—Scaling trends of reconfigurable hardware (RH) and their design flexibility have proliferated their use in dependability-critical embedded applications. Although their reconfigurability can enable significant fault tolerance, due to the complexity of execution time in their design flow, in-field reconfigurability can be infeasible and thus limit such potential. This need is addressed by developing a graph and set theoretic approach, named *hypergraph-cover diversity* (HCD), as a preemptive design technique to shift the dominant costs of resiliency to design-time. In particular, *union-free hypergraphs* are exploited to partition the reconfigurable resources pool into highly separable subsets of resources, each of which can be utilized by the same synthesized application netlist. The diverse implementations provide reconfiguration-based resilience throughout the system lifetime while avoiding the significant overheads associated with runtime placement and routing phases. Two novel scalable algorithms to construct union-free hypergraphs are proposed and described. Evaluation on a Motion-JPEG image compression core using a Xilinx 7-series-based FPGA hardware platform demonstrates a statistically significant increase in fault tolerance and area efficiency when using proposed work compared to commonly-used modular redundancy approaches.**

I. Introduction

The exponential growth in number of switching devices in very-large-scale integration (VLSI) designs dictated by Moore's law has rapidly increased the likelihood of fault occurrence in hardware systems. This challenge is aggravated further by the advent of nanofabrication technology which introduced unprecedented reliability issues. Thus, *fault tolerance* (FT) techniques for computing systems have received a considerable attention in recent years. Traditionally, FT techniques have relied on passive modular redundancy which have a limited benefit per unit area and power. For many critical embedded systems such as those used in space and avionics platforms, unmanned aerial vehicles (UAVs), land and ocean-based remote sensing units (RSUs), environmental stress and conditions can result in a failure rate beyond what fault-handling capability of passive approaches can resolve. For such applications, immediate hands-on maintenance and repair is infeasible and hence a duly deployed autonomous method which caters to fault tolerance using available healthy resources is crucial for maintaining reliable long-term operation.

With the rise of *reconfigurable hardware* (RH) over the last two decades, in-field reconfigurability has opened up new possibilities to incorporate pseudo-intelligent FT attributes such as self-repair and autonomous fault recovery [1]. Such attributes are key enablers for efficient and sustainable fault-tolerant systems. RH is expected to have an essential role in designing future dependable embedded systems [2]. Unfortunately, exploiting design flexibility of modern RH for runtime FT is encumbered by the heuristic nature and increasing complexity of design placement and routing mechanisms. *SRAM-based field programmable gate arrays* (FPGAs) being the prominent example of RH can exemplify this challenge. Execution of a design flow targeting an SRAM-based FPGA can take an order of minutes to hours using a high-end multi-processing machine [3]. For low-performance fabric-embedded cores, the computational and energy constraints to execute in-field design reroute can be prohibitive [4]. We emphasize our observation here based on the current state of computer-aided design (CAD) tools used with available commercial off-the-shelf (COTS) reconfigurable components due to the increasing trend in using COTS-based embedded systems [5] [6].

In light of this major concern, design-time FT strategies that minimize reliance on runtime execution of design flow are sought which can be readily integrated with existing vendor tools. More specifically, the dominant implementation cost of reconfigurability feature can be mitigated by preparing an optimal set of design alternatives at design phase that properly cover the solution space for reliability exposures at runtime. In this work, *hypergraph-cover diversity* (HCD) approach based on *graph and set theory* is proposed to attain this objective for the FT coverage problem on embedded reconfigurable fabric. The HCD method exploits the strong notion of *separability* [7] obtained by *union-free hypergraphs* [8] to model resource allocation among distinct design alternatives for highly diverse and fault resilient designs.

The remainder of the paper is organized as follows. Section II provides background and summarizes related work. Section III describes the graph model for proposed work. Section IV discuses the case study adopted for evaluation and experimental results. Finally, conclusions are given in Section V.

II. Background and Related Work

FPGA-based embedded systems have become ubiquitous computing platforms owing to the increasing embedded system functionality and the low non-recurring engineering costs (NREs) of embedded processor development using FPGAs. A typical FPGA-based embedded system may comprise application specific integrated circuits (ASICs) blocks, e.g. digital signal processors (DSPs), peripheral interfaces, memory controllers to interface with external non-volatile and main

memory, a system bus, and a reconfigurable fabric tightly coupled to a general purpose processor (GPP). The reconfigurable fabric can act as a general hardware accelerator for performance-critical functions or as a flexible design circuitry to apply runtime changes such as protocol extensions, bug fixes, or advanced features to existing implementation [9]. Resources in the reconfigurable fabric are organized in a regular 2D array of identical tiles. Each tile includes a single configurable logic block (CLB) as well as switching and connection blocks to facilitate inter and intra CLB connections. A CLB is also referred to as a logic array block (LAB) depending on FPGA device vendor. Each CLB is a group of identical programmable logical cells called slices. A slice can have several look-up tables (LUTs), flip-flops (FFs), multiplexers, carry chains, and dedicated gates for combining LUTs to realize more complex Boolean functions.

A. Previous Work on FT on Reconfigurable Hardware

The regularity and logic density of contemporary reconfigurable architectures are particularity well suited for provision of runtime FT. A great deal of research has been conducted in the area of FT of reconfigurable hardware [10] [11]. Most FT approaches that exploit runtime reconfiguration for fault recovery can be classified as *covering approaches* or *embedding approaches* [12].

In the *covering approaches*, a set of spare resources are predefined and made available to replace faulty elements [1] [13]. The reconfiguration problem is to find the optimal assignment of spares to faulty resources such that large combinations of faulty resources can be covered. In [1], assignment of spare columns of resources is proposed for single-fault tolerance. A faulty column is avoided by shifting the column-based design implementation to a different set of healthy columns. Since number of spare assignments is low, they are implemented at design-time and used during system lifetime to provide low-overhead fault recovery. A single faulty resource in a column renders the whole column unusable and hence this method can result in a low area efficiency. The work in [13] distributes locations of individual spare resources evenly across the fabric boundary. Distance-based evaluation score is used to define spare assignment at runtime and incremental runtime re-routing is required to achieve fault recovery. Although, this technique can cover a large set of multiple faults, its practicality can be very challenging given the complexity and routing overhead of contemporary FPGAs.

Alternatively, *embedding approaches* make no distinction between spare and normal resources. A system should have more resources than what is required to implement an application. Fault tolerance is achieved by remapping (embedding) fault-effected design into (in) remaining healthy resources. The challenge in this approach is to define a minimal set of alternative designs that achieve the target level of fault tolerance. A heuristic search algorithm to generate a set of diverse designs for single-fault tolerance at the CLB level was recently proposed in [14]. To the best of our knowledge, no formal technique based on a theoretical concept has been proposed as an embedding approach to real-time FT in reconfigurable systems. In this paper, we exploit the notion of set separability given by the union-free hypergraphs [8] to identify highly diverse and fault resilient designs.

A different hypergraph model was previously used in [15] to define a FT connection topology as a yield enhancement technique for processor arrays. Hypergraphs were also used to study the spare assignment problem in the covering approach of FT processor arrays [16]. In [17], a graph-based software technique to lower complexity of real-time reconfiguration problems in distributed service-based systems is discussed. Although software-based techniques can be helpful to extract the optimal software-hardware mapping to enable dynamic real-time reconfiguration, they do not address the cases where reconfiguration overhead is largely determined by the reconfiguration procedure of the underlying hardware. A design-time approach to realize static configurations using design disjunction was recently proposed in [18] to address real-time fault diagnosis and recovery problems without elaboration on how the disjunct designs are created. The following subsection will describe the proposed graph model developed herein with examples.

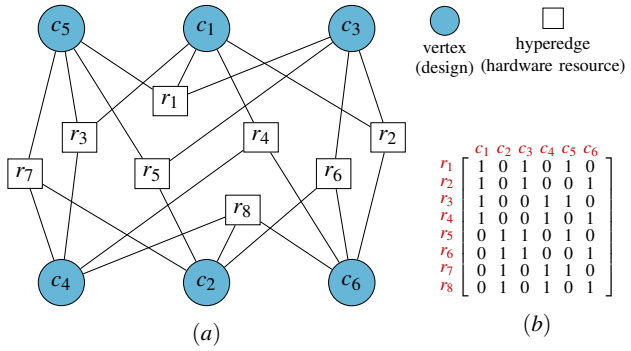
B. Union-free Hypergraphs

A hypergraph $H = (V\ ,\ E_h)$ can be described as a generalization of a graph in which edges E_h, or hyperedges, can connect any number of vertices. For the interest of this work, resources will be represented by hyperedges and vertices are considered distinct designs. Thus, a hypergraph in this representation defines how resources (hyperedges) are allocated to (connect to) different designs (vertices). Each hyperedge can be expressed by the subset of vertices it connects to. In hypergraph theory, a highly strong notion of set separability is described by a class of hypergraphs known as union-free hypergraphs. Consider he_x, he_y, and he_z to be three distinct hyperedges in H. Based on the original definition of union-free property [19], H is union-free if:

$$\forall\ he_x,\ he_y,\ he_z \in E_h\ ,\ \ he_x \cup he_y \neq\ he_z \tag{1}$$

This definition implies that there are no two distinct hyperedges (resources) in H connected to (utilized by) the same set of vertices (designs). Fig. 1(a) shows an example of a union-free hypergraph with six vertices and eight hyperedges and its incidence matrix in Fig. 1(b). Columns of the incidence matrix represent hyperedges (resources) and rows represent vertices (designs). An element x_{ij} of the incidence matrix is one if hyperedge i connects to vertex j. Therefore, non-zero elements in each column define the subset of resources used by each design.

We propose a systematic way for constructing such a union-free hypergraph using trivial binary matrix manipulations. Given that embedding approaches require more resources than what are needed for design implementation, the *resource utilization ratio U* is expected to be less than 1. In the next



(a) (b)

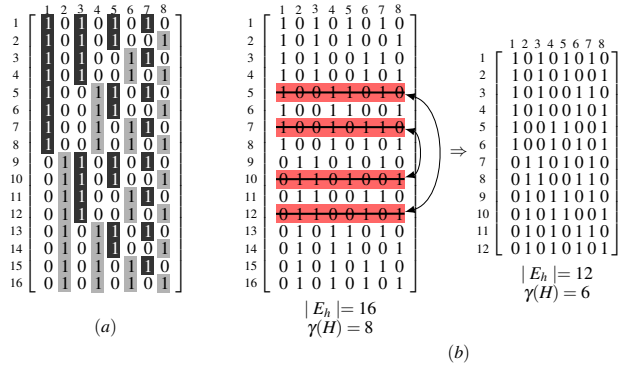
Fig. 1: Example of a (a) Union-free Hypergraph and (b) its Incidence Matrix

section, we describe two novel algorithms to construct the described hypergraph model for utilization ratio $U = 1/2$ and $U = 2/3$. These target values result in a low area overhead compared to commonly-used modular redundancy schemes, e.g. Triple Modular Redundancy (TMR).

III. Hypergraph-Cover Diversity

To construct a union-free hypergraph H suitable for generating resilient designs, two properties have to be maintained:

- **First**, condition (1) must be satisfied.
- **Second**, H is a regular[1] hypergraph with a degree[2] $\gamma(H) = A$, where A is the minimum number of resources required to implement the application.



(a) (b)

Fig. 2: (a) Example of Union-free Hypergraph Construction using Incidence Matrix for $U = 1/2$ (b) Reducing Degree of Hypergraph by Deleting Disjoint Hyperedges

We propose the first algorithm, Algorithm 1, for $U = \frac{1}{2}$ which achieves union-free property using only $\lceil 2 \cdot log_2(2 \cdot A) \rceil$ designs. For the sake of clarity, the incidence matrix is used for the illustration of the proposed algorithm. The corresponding steps in the graph domain are given in the pseudo-code. As an example, assume the size of the target application is $A = 6$ FPGA slices, the algorithm starts with a zero incidence matrix having a number of rows $r = 2^x$, where $x = \lceil log_2(2 \cdot A) \rceil$, thus $r = 16$, and a number of columns equals $c = 2 \cdot x$, hence $c = 8$, (lines 1 through 7 of Algorithm 1). For odd-indexed columns $(1, 3, ..., r-1)$, the elements indicated by 'one' are distributed according to the binary tree pattern depicted in Fig. 2(a). Each of the even-indexed columns $(2, ..., r)$ is obtained by simply complementing its immediate lower-indexed column. These steps are more formally defined in lines 10 through 21 of the pseudo-code.

The hypergraph given by the resultant incidence matrix should satisfy condition (1). A further step is needed to reduce the degree of the graph to $\gamma(H) = A$. We observe that complimentary rows (or disjoint hyperedges) can be deleted without violating condition (1). Deleting a pair of disjoint hyperedges will decrement $\gamma(H)$ by 1 as shown in Fig. 2(b). This step is repeated until $\gamma(H) = A$ (lines 22 through 28 of Algorithm 1).

Algorithm 1: Algorithm for Constructing Union-free Hypergraph for $U = 1/2$

Procedure construct hypergraph $H = (V, E_h)$
Input: A: Resource count required to implement application
Output: Hypergraph $H = (V, E_h)$

```
1  V := ∅ // set of vertices
2  E_h := ∅ // set of hyperedges
3  x := ⌈log2(2·A)⌉
4  r := 2^x // initial number of hyperedges
5  c := 2·x // initial number of vertices
6  V := {v1, v2, ...., vc} // initialize V with c vertices
7  E_h := {he1, he2, ...., her} // initialize E_h with r hyperedges
8  k := 0 // level index in the binary tree
9  c_index := 1 // index to loop through each pair of vertices
10 while (c_index < c) do
11     p := 2^k // parameter to determine number of leaves in
           each level of the binary tree
12     e := r/(2·p) // parameter to determine subset of
           hyperedges connected to each pair of vertices
13     i := 1 // index to loop through all r hyperedges
14     while (i < r) do
15         for m := i → (i + e) do
16             he_m := he_m ∪ {v_{2·k+1}} // inclusion of
                   odd-indexed vertex
17         for m := i + e → (i + 2·e) do
18             he_m := he_m ∪ {v_{2·k+2}} // inclusion of
                   even-indexed vertex
19         i := i + 2·e
20     k := k + 1
21     c_index := c_index + 2
   // reduce degree of hypergraph H by deleting disjoint
       hyperedges, one pair at a time
22 if γ(H) ≠ A then
23     for every he_x, he_y ∈ E_h | x ≠ y do
24         if he_x ∩ he_y = ∅ then
25             remove he_x and he_y from H
26             if γ(H) = A then
27                 return H
28                 end
```

In a similar manner, an algorithm can be devised for hardware utilization $U = \frac{2}{3}$. Algorithm 2 achieves union-free

[1]A *regular* hypergraph is a graph where each vertex has the same number of edges

[2]The *degree* of a regular graph is the number of edges connected to each vertex

resource assignment using $\lceil 3 \cdot log_3(\frac{3}{2} \cdot A) \rceil$ designs. Assuming $A = 5$, the algorithm starts with a zero incidence matrix having a number or rows $r = 3^x$, where $x = \lceil log_3(\frac{3}{2} \cdot A) \rceil$, thus $r = 9$, and a number of columns equals $c = 3 \cdot x = 6$, as given in lines 1 through 7 of Algorithm 2.

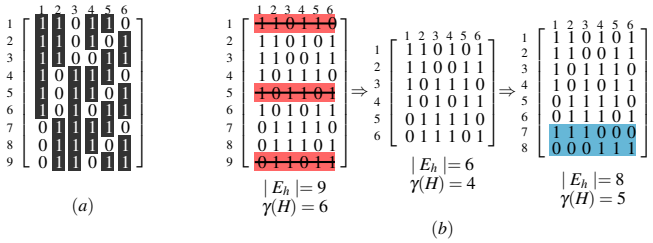


Fig. 3: (a) Example of Union-free Hypergraph Construction using Incidence Matrix for $U = 2/3$ (b) Adjusting Degree of Hypergraph to Fit Target Application Size

The distribution of the one-encoded elements in the incidence matrix follows a ternary tree pattern among three groups of columns: (1,4,7,..., etc.), (2,5,8,..., etc.), and (3,6,9,..., etc.). Since $U = 2/3$, two thirds of each column's elements must be non-zero. As shown in Fig. 3(a), the order of those thirds, from top to bottom, is (1^{st}, 2^{nd}), (1^{st}, 3^{rd}), and (2^{nd}, 3^{rd}) for the three groups of columns, respectively (lines 10 through 23 of Algorithm 2). This arrangement results in a union-free hypergraph in which $\gamma(H) = \frac{2}{3} \cdot r$.

By removing hyperedges he_x, he_y, he_z such that:

$$|he_x \cap he_y| = |he_x \cap he_z| = |he_y \cap he_z| = 2$$

$\gamma(H)$ is decremented by two while preserving the union-free property (lines 24 through 30). If A is an odd number, a last step to pad the incidence matrix is necessary to reach $\gamma(H) = A$. Padding is conducted by simply adding a pair of disjoint hyperedges (lines 31 through 36 of Algorithm 2) or complementary rows as shown in Fig. 3(b) to attain $A = 5$.

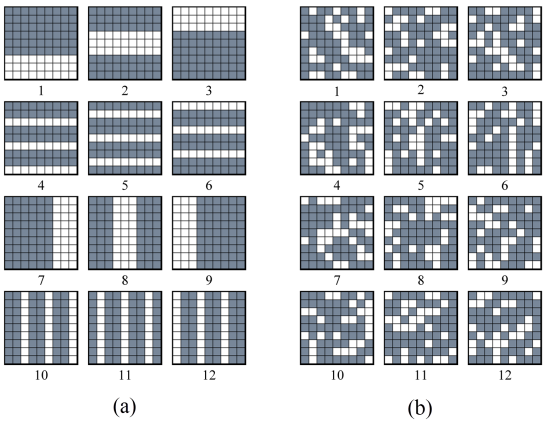


Fig. 4: Two Equivalent Sets of Separable HCD Resource Allocations on 2D Array

A salient attribute of the proposed HCD technique that is crucial to its practicality and effectiveness is the ability to swap

Algorithm 2: Algorithm for Constructing Union-free Hypergraph for $U = 2/3$

Procedure construct hypergraph $H = (V\ ,\ E_h)$
Input: A: Resource count required to implement application
Output: Hypergraph $H = (V\ ,\ E_h)$

1 $V := \emptyset$ // set of vertices
2 $E_h := \emptyset$ // set of hyperedges
3 $x := \lceil log_3(\frac{3}{2} \cdot A) \rceil$
4 $r := 3^x$ // initial number of hyperedges
5 $c := 3 \cdot x$ // initial number of vertices
6 $V := \{v_1, v_2, ..., v_c\}$ // initial number of vertices
7 $E_h := \{he_1, he_2, ..., he_r\}$ // initialize E_h with r hyperedges
8 $k := 0$ // level index in the ternary tree
9 $c_{index} := 1$ // index to loop through each triad of vertices
10 **while** $(c_{index} < c)$ **do**
11 $p := 3^k$ // parameter to determine number of leaves in each level of the ternary tree
12 $e := \frac{r}{3 \cdot p}$ // parameter to determine subset of hyperedges connected to each triad of vertices
13 $i := 1$
14 **while** $(i < r)$ **do**
15 **for** $m := i \rightarrow (i + e)$ **do**
16 $he_m := he_m \cup \{v_{3 \cdot k+1}, v_{3 \cdot k+2}\}$ // inclusion of 1^{st} & 2^{nd} vertices of each triad
17 **for** $m := i + e \rightarrow (i + 2 \cdot e)$ **do**
18 $he_m := he_m \cup \{v_{3 \cdot k+1}, v_{3 \cdot k+3}\}$ // inclusion of 1^{st} & 3^{rd} vertices of each triad
19 **for** $m := i + 2 \cdot e \rightarrow (i + 3 \cdot e)$ **do**
20 $he_m := he_m \cup \{v_{3 \cdot k+2}, v_{3 \cdot k+3}\}$ // inclusion of 2^{nd} & 3^{rd} vertices of each triad
21 $i := i + 3 \cdot e$
22 $k := k + 1$
23 $c_{index} := c_{index} + 3$
// reduce degree of hypergraph H by deleting appropriate hyperedges, one triad at a time
24 **if** $\gamma(H) \neq A$ **then**
25 **for** $he_x, he_y, he_z \in E_h \mid x \neq y \neq z$ **do**
26 **if** $|he_x \cap he_y| = |he_x \cap he_z| = |he_y \cap he_z| = 2$ **then**
27 remove he_x, he_y, and he_z from H
28 **if** $\gamma(H) = A$ **then**
29 return H
30 end
31 **if** $\gamma(H) < A$ **then**
// add a one pair of disjoint hyperedges to attain $\gamma(H) = A$
32 $he_1 := \{\lceil \frac{c}{2} \rceil$ random vertices from $V\}$
33 $he_2 := \{v \mid v \notin he_1\}$
34 add he_1 and he_2 to E_h
35 return H
36 end

all hyperedges in the hypergraph without violating condition (1) or degrading the achieved separability of resultant designs. Fig. 4 depicts how resources are assigned to diverse designs using a union-free hypergraph for a reconfigurable region of size $9 \cdot 9 = 81$ and application size $A = 54$. Algorithm 2 is used in this example to define 12 separable designs. The 81 resources are ordered from left to right and up to down. Fig. 4(a) shows the arrangement of resources as defined by Algorithm 2 without changing the order of hyperedges. Fig. 4(b) shows how this arrangement is mapped to an equivalent one by randomly swapping all hyperedges of the constructed hypergraph. Both

arrangements exhibit the same resource separability dictated by the union-free property. This feature is important because routing congestion constraints of tightly-closed resources, as in Designs: 1,3,7, and 9 of Fig. 4(a), can lead to infeasible routing. In addition, timing violations can occur if distant groups of resources, as given by Designs: 2 and 8 of Fig. 4(a), are used to implement design alternatives. The number of distinct arrangements that can be constructed is intractably vast which necessitates the use of CAD techniques for finding the optimal performance and power consumption goals while achieving reliability goals.

IV. Evaluation

A discrete cosine transform (DCT)-based motion-JPEG (MJPEG) compression core was selected as a case study for this work due to its widespread use for image and video compression. Image and video compression applications are commonly used in space exploration missions, satellites, and monitoring systems in high radiation-dose and harsh environments. Such systems require rigid requirements in system reliability and durability. This includes the ability to operate for a long lifetime while providing adequate processing to meet increasing future demands. For instance, systems used in low-orbit satellites are expected to remain in service for up to 25 years [20]. A failure in an on-board system can render a mission objective obsolete. For survivable systems under these conditions where hands-on human maintenance is infeasible, a form of redundancy is mandatory to replace or mask failed components.

Evaluation of HCD design approach is conducted on an FPGA-based computing system using the commercial Xilinx KC705 FPGA evaluation board [21]. The KC705 board includes a 28nm Xilinx 7-series FPGA, DDR3 and FLASH memory. Xilinx's embedded development kit (EDK) and PlanAhead tools are used to generate the bitfiles for all of the implementations in this evaluation. Bitfiles are stored in an external flash memory chip before evaluation procedures begin. An embedded MicroBlaze soft processor is employed to monitor system correctness and control reconfiguration flow for autonomous fault recovery when a failure is detected. We use Xilinx's software development kit (SDK) to develop the software module which runs on the embedded processor to reconfigure HCD designs onto the device and to control the FT flow. The processor can be protected against soft and hard faults using proper techniques such as TMR, rollback, checkpointing, or reconfiguration-based approaches [22] [23]. Partial reconfiguration using Xilinx's internal configuration access port (ICAP) is utilized in this work for rapid reconfiguration.

The MJPEG core comprises different blocks, most of which are not computationally intensive and, hence, are implemented by software processing using the embedded MicroBlaze processor. Contrarily, the 2D-DCT block of the MJPEG core imposes a high computational workload as reflected by the latency ratio of all blocks in the left stacked bar of Fig. 5(a). To reach balanced critical paths among MJPEG pipeline stages, it is favorable to implement DCT block as a hardware accelerator on reconfigurable fabric. Fig. 5(b) shows the normalized latency of DCT implementation using software and hardware processing on the KC705 board. The right stacked bar of Fig. 5(a) shows modified latency ratios of MJPEG blocks after a DCT accelerator is implemented on reconfigurable fabric. The proposed HCD scheme is applied to the DCT hardware accelerator. Resource assignments of HCD designs are defined at the slice level and enforced using placement constraints applied by inserting the placement `AREA_GROUP` and `CONFIG_PROHIBIT` statements in the user constraints file (UCF) used by the Xilinx place-and-route (PAR) tool.

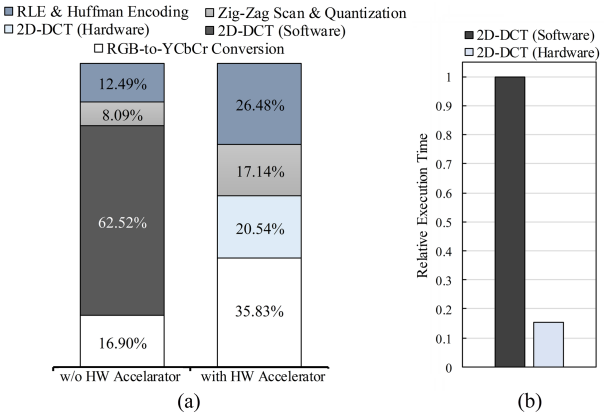


Fig. 5: Latency Ratio of MJPEG Blocks on the KC705 Evaluation Board

By using a quantifiable image quality metric such as the peak signal-to-noise ratio (PSNR), the need for hardware error detection can be eliminated, thereby a large saving in area and power is achieved [24]. PSNR is based on mean square error (MSE) computed by averaging the intensity difference between pixels of recovered and original images. The recovered image is obtained by a fast software-based implementation of the inverse discrete cosine transform (IDCT). The input test image in this evaluation is a high-resolution satellite image loaded to the on-board DDR3-DRAM memory before the evaluation is conducted.

TABLE I: Design Parameters for Implemented TMR and HCD

Technique	Design Alternatives	Power Consumption (mW)		Resource Count (slices)	
Hypergraph-Cover Diversity	18	Min.	23.29	Total	844
		Avg.	24.11	Active Resources	422
		Max.	25.40		
TMR	1	71.01		Total	1350
				Active Resources	1270

In this evaluation, HCD is compared to the most frequently used FT technique, TMR, which is the conventional strategy used by the Xilinx X-TMR FT tool. Table I shows the resource count and power consumption of a DCT accelerator for HCD and TMR implementations. 422 slices are required to implement a single DCT accelerator. Algorithm 1 is used to generate HCD alternatives for hardware utilization $U = \frac{1}{2}$; hence total number of resources is 844 slices. Only 18 design alternatives

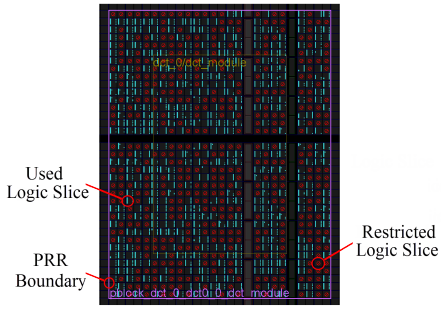


Fig. 6: PlanAhead Layout for a Single HCD Design Alternative

were found sufficient to achieve union-free property. Fig. 6 shows a screenshot of the PlanAhead tool layout of the partial reconfiguration region (PRR) for an HCD implementation of a DCT accelerator.

Since FPGA routing resources are configured to realize the structure of target design using distinct sets of logic resources, we would expect a low utilization overlap of routing resources among HCD designs. Fig. 7 shows the proportion of all used programmable interconnect points (PIPs) according to their utilization count among HCD designs for the DCT implementation. We observe a steep exponential decline in the number of PIPs that are utilized by more than one design. This observed utilization rate indicates fault resilience of the proposed method against routing failures as evaluated herein.

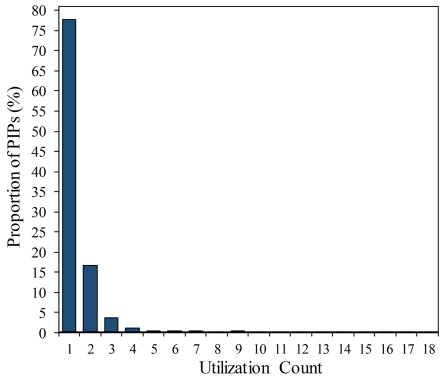


Fig. 7: PIPs Utilization Overlap among HCD Designs

For TMR implementation, three replicas of a DCT accelerator with a voting logic are realized using $3 \cdot 422 + 4$ (for voter) $= 1270$ slices. For each replica, an extra 28 unused slices are found to be required to enable feasible routing. This issue does not affect the implementation of HCD designs since half of the total resources in the PRR must be unused.

To compare effectiveness of fault tolerance of HCD and TMR methods, up to 40 stuck-at faults (SAFs) at LUTs input terminals are randomly injected based on a uniform distribution. Input terminals are selected for fault injection to capture the effect of both logic and interconnect faults. Fig. 8 shows the averaged retrieved PSNR values over 25 experimental runs for HCD and TMR schemes. Due to the

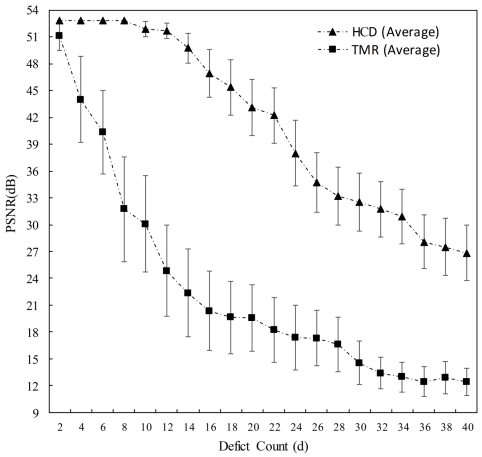


Fig. 8: PSNR Results for TMR and Proposed HCD Implementations

large set of possible combinations of fault locations that can be injected, the upper and lower whiskers indicating the 95% confidence interval of the results are provided at each evaluation point. A PSNR value for the HCD implementation represents the highest PSNR found by the embedded processor after evaluating all 18 HCD alternatives. The proposed FT scheme maintains a substantially higher image quality throughout the evaluation. It is observed that image quality of TMR design deteriorates rapidly as the defect count d becomes larger than one. Contrastingly, the HCD approach maintains a minimal degradation in output quality for up to 12 faults and a lower deterioration rate under a higher defect count with roughly twice the image quality of TMR.

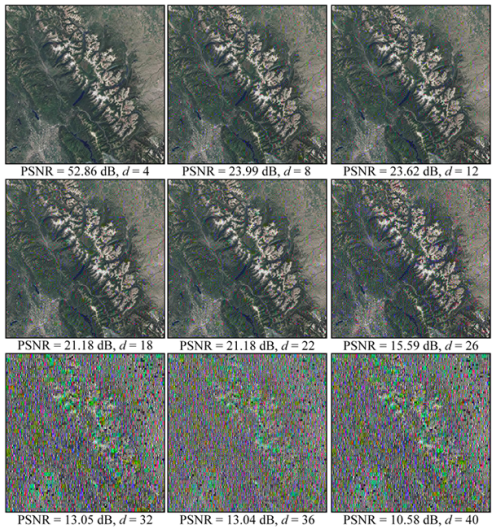


Fig. 9: Image Quality Comparison under Varying Defect Count using TMR Scheme

Fig. 9 and Fig. 10 show output JPEG images for an experimental run using TMR and HCD methods, respectively. The results here depict the considerable advantage of HCD

to tolerate faults compared to static modular redundancy. The effect of hardware defects on image quality of TMR design can be apparent to human perception for $d > 8$, whereas in the case of HCD implementation, degradation in image quality is hardly distinguishable even for $d = 40$.

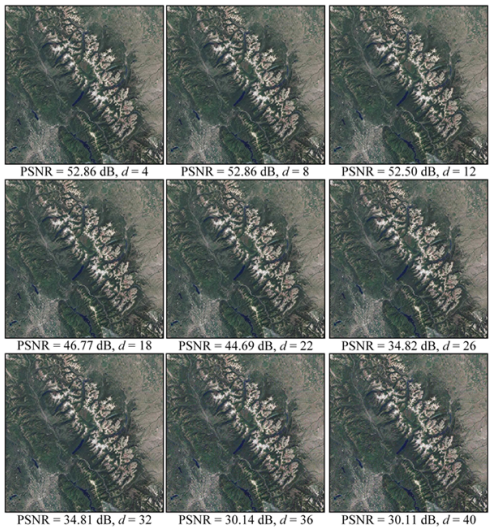


Fig. 10: Image Quality Comparison under Varying Defect Count using HCD Approach

Table I also indicates the area and power measurements of the two implementations. The proposed work results in a total saving of 37.5% compared to TMR when weighed against their increased area overhead. The average power consumption of HCD design alternatives is 24.11 mW. Compared to TMR, HCD achieves a power saving of 66%. This is attributed to the fact that only one DCT accelerator is needed in the case of HCD to provide fault tolerance as opposed to the three replicas of the TMR scheme.

V. Conclusion

In this work, we show that set separability defined by hypergraph theory can be used effectively to create highly resilient designs at design-time for provision of low-overhead fault recovery during system lifetime. Systematic algorithms to construct design diversity using the union-free hypergraph model for different target hardware utilization values are also presented. Results have demonstrated the potential of the proposed FT method to achieve 37.5% area saving and up to 66% reduction in power consumption compared to the frequently-used TMR scheme while providing superior fault tolerance.

Acknowledgment

This research has been supported by the Ministry of Higher Education of Saudi Arabia under scholarship grant no. 64923.

References

[1] S. Mitra, W.-J. Huang, N. R. Saxena, S.-Y. Yu, and E. J. McCluskey, "Reconfigurable architecture for autonomous self-repair," *IEEE Des. Test. Comput.*, vol. 21, no. 3, pp. 228–240, Jun. 2004.

[2] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Hartig, L. Hedrich *et al.*, "Design and architectures for dependable embedded systems," in *Proc. of IEEE 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'11)*, Taipei, Taiwan, Oct. 2011, pp. 69–78.

[3] T. Feist, "Vivado design suite," Xilinx, White Paper WP416 (v1.1), Jun. 2012.

[4] W. Zha, "Facilitating FPGA reconfiguration through low-level manipulation," Ph.D. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA, Feb. 2014.

[5] M. Pignol, "COTS-based applications in space avionics," in *Proc. of Design, Automation and Test in Europe Conference (DATE'10)*, Dresden, Germany, Mar. 2010, pp. 1213–1219.

[6] R. Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu, "Hardware runtime monitoring for dependable COTS-based real-time embedded systems," in *Proc. of IEEE 29th Real-Time Systems Symposium (RTSS'08)*, Barcelona, Spain, Nov./Dec. 2008, pp. 481–491.

[7] B. Bollobás, *Modern graph theory*. Springer Science & Business Media, 1998, vol. 184.

[8] P. Frankl and Z. Föredi, "Union-free hypergraphs and probability theory," *European Journal of Combinatorics*, vol. 5, no. 2, pp. 127–131, 1984.

[9] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2006, no. 1, pp. 13–13, Jan. 2006.

[10] E. Stott, P. Sedcole, and P. Cheung, "Fault tolerance and reliability in field-programmable gate arrays," *IET Computers & Digital Techniques*, vol. 4, no. 3, pp. 196–210, May 2010.

[11] M. G. Parris, C. A. Sharma, and R. F. Demara, "Progress in autonomous fault recovery of field programmable gate arrays," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 31, Oct. 2011.

[12] R. Libeskind-Hadas, N. Hasan, J. Cong, P. K. McKinley, and C. L. Liu, *Fault Covering Problems in Reconfigurable VLSI Systems*. Norwell, MA: Kluwer Academic Publishers, 1992.

[13] J. Emmert, C. Stroud, and M. Abramovici, "Online fault tolerance for fpga logic blocks," *IEEE Trans. VLSI Syst.*, vol. 15, no. 2, pp. 216–226, Feb. 2007.

[14] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures," in *Proc. of IEEE International Test Conference (ITC'13)*, Anaheim, CA, Sep. 2013, pp. 1–10.

[15] A. L. Rosenberg, "A hypergraph model for fault-tolerant VLSI processor arrays," *IEEE Trans. Comput.*, vol. C-34, no. 6, pp. 578–584, Jun. 1985.

[16] K. Sugihara and T. Kikuno, "On fault tolerance of reconfigurable arrays using spare processors," in *Proc. of IEEE Pacific Rim International Symposium on Fault Tolerant Systems (PRFTS'91)*, Kawasaki, Japan, Sep. 1991, pp. 10–15.

[17] M. Garca-Valls, P. Uriol-Resuela, F. Ibez-Vzquez, and P. Basanta-Val, "Low complexity reconfiguration for real-time data-intensive service-oriented applications," *Future Generation Computer Systems*, vol. 37, pp. 191–200, Jul. 2014.

[18] A. Alzahrani and R. F. DeMara, "Non-adaptive sparse recovery and fault evasion using disjunct design configurations," in *Proc. of ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA'14)*, Monterey, CA, Feb. 2014, pp. 251–251.

[19] P. Erdös and S. Shelah, "On problems of moser and hanson," in *Graph theory and applications*. Springer, 1972, vol. 303, pp. 75–79.

[20] "Process for limiting orbital debris," NASA, Technical Standard NASA-STD-8719.14A, Dec. 2011. [Online]. Available: http://http://www.hq.nasa.gov/office/codeq/doctree/871914.pdf

[21] "KC705 evaluation board for the Kintex-7 FPGA," Xilinx, User Guide UG810(v1.6.1), Apr. 2015.

[22] A. Vavousis, A. Apostolakis, and M. Psarakis, "A fault tolerant approach for FPGA embedded processors based on runtime partial reconfiguration," *Journal of Electronic Testing: Theory and Applications*, vol. 29, no. 6, pp. 805–823, Dec. 2013.

[23] M. Violante, C. Meinhardt, R. Reis, and M. Reorda, "A low-cost solution for deploying processor cores in harsh environments," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 2617–2626, Jul. 2011.

[24] G. Varatkar and N. Shanbhag, "Error-resilient motion estimation architecture," *IEEE Trans. VLSI Syst.*, vol. 16, no. 10, pp. 1399–1412, Oct. 2008.