Fault Demotion Using Reconfigurable Slacks (FADRES)

Abstract—**We propose a novel active dynamic redundancy based fault handling approach exploiting the partial dynamic reconfiguration capability of FPGAs. Our first contribution is error detection in uniplex functions thereby avoiding any redundancy during normal operation. Second, we present a novel fault isolation scheme which neither requires test vectors nor disrupts the normal throughput of the systems. Third, the deterministic flow of the fault isolation scheme guarantees success in bounded number of reconfigurations. The approach is validated by taking H.263 Video encoder's DCT block as a case study. The PSNR measure of the video sequences shows that we are able to achieve fault tolerance in the DCT block with only minor quality degradation during the fault isolation phase spanning a few frames.**

Keywords-**fault tolerance; partial reconfiguration; fault handling by reconfiguration**

I. Introduction

With today's rapidly scaling technology, the reliability issue is posing a clear challenge to the correct operations of modern complex computing systems. Fault tolerance of mission critical systems is of utmost importance. In some situations, even sustainability in the form of graceful degraded mode is desired if the full recovery cannot be guaranteed. The reconfiguration capability of FPGAs is promising for building autonomous fault handling systems [1], [2], [3]. On the other hand, FPGAs are also susceptible to soft (transient) errors as well as hard (permanent) faults. Therefore, the importance of addressing reliability issues is growing, especially in this era of rapidly improving semiconductor technology in terms of integration level, performance, and transistors density on chip.

Reconfigurable hardware fabric has been widely used as a platform for signal processing applications. The inherently parallel architecture of FPGAs is very benefecial for time demanding tasks, such as image/video coding applications, cryptographic algorithms, and speech processing. Current state of the art video coding technologies such as H.264, are highly effective for compressing video sequences. However, due to this high compression ratio, image/video coding applications are extremely vulnerable to errors. Traditionally, various techniques of error resilience have been developed by adding redundancy into the compressed bitstream at the source or channel coder stages. However, there is a scarce of research dealing with the hardware faults of these applications. Reliability problem of highly complex VLSI systems in nanometer process, caused by soft and hard errors, is increasing. The induced faults during these coding processes will lead to disastrous impact on the output of highly compressed bitstream.

An obviously effective approach to handle faults of permanent nature in FPGA is to diagnose fault and then functional block can be reconfigured to utilize fault-free hardware resources. The partial reconfiguration (PR) ability of FPGA is useful in systems requiring run-time adaptability [4]. In case of fault, the system is desired to continue its operation without stopping its service. An online fault handling scheme would not interrupt the continuous throughput of the system, while temporary performance degradation in terms of spatial/temporal resolution, or *Peak Signal-to-Noise Ratio* (PSNR) quality, might be necessary. However, minimal impact on PSNR is desirable with fault recovery phase completion as soon as possible.

In our approach, we employ redundancy to isolate and recover from faults, thereby avoiding exhaustive test vectors to test the functions configured on hardware fabric. Once the proposed system detects fault, the fault isolation phase is initiated by reconfiguring multiple *Processing Elements* (PEs) with the same function for discrepancy check. The dynamic reconfiguration property of FPGA is employed to create the redundancy needed for the fault isolation process during run-time. If there is no redundant PE available, the lower priority functional module can be made free. The outputs from these *Reconfigurable slacks* are concurrently checked for discrepancy with that of functional module in the data path providing normal throughput. The discrepancy in output of the same functional modules reveals permanent or transient fault. The proposed algorithm iteratively evaluates the functional modules keeping them in the main data path, as well as the checking slacks.

The real time analysis of time varying characteristics of input data is helpful in predicting the computational complexity and hence the required hardware resources. Hardware architecture with software flexibility is desirable to provide architectural support to deal with these time-varying computing workloads. Then, the hardware resources saved by intelligent prediction of computational resources are used to provide the capability needed for the proposed fault isolation and recovery scheme. The simulation results show that fault isolation can be improved by taking into account certain parameters (e.g. *Quantization Parameter* QP) and input signal characteristics (e.g. motion activity in the scene).

Table I
COMPARISON OF FAULT TOLERANCE TECHNIQUES FOR FPGAS

Approach	**Area requirement**	**Basis for Recovery**	**Detection Latency**	**Number of Reconfigs**	**Additional Components Required**	**Granularity**	**Guarantee of improvement**
TMR	3 times	Requires 2 datapaths are operational	Negligible	Not Reconfigurable	2 of 3 Majority Voter	Coarse	100% for single fault, 0% thereafter
Evolutionary Techniques	NA	Redundancy during design	NA	Non-deterministic	None at run-time	Circuit Level	No
Online Recovery (Roving STARs, Online BIST) [5], [6]	Roving Area	Available Spares	Varies	Varies	Test vector generator, Output response analyzer	-	Yes
CRR [7]	Duplex	Recovery complexity	Negligible	Varies	CRR controller	Resource Level	No
FADRES (proposed herein)	Uniplex	Priority Functions	Negligible	Bounded	FADRES FSM	Algorithm Level	Yes

II. RELATED WORK

The redundancy based methods are popular among fault tolerant systems community, with side effects of area and power overhead. On the other hand, test vector strategy isolates faults with small area overhead, yet at the cost of huge number of test vectors to diagnose the functional blocks as they increase exponentially according to the number of inputs. Our active dynamic redundancy approach enjoys the benefits of redundancy with negligible computational overhead. The static redundancy techniques usually reserve spares for fault handling. In contrast, in our work the redundant modules are active as they are part of the system during normal operation, performing their assigned tasks. We introduce redundancy only if needed, during run-time.

Table I provides a comparison of previous approaches towards fault handling in FPGA based systems. The TMR technique involves triplication of the design where the three copies of the systems are running all the time. The fault capacity is limited to the faults within one copy only. Evolutionary techniques repair the circuit at finer granularity, but there is no guarantee that the solution would be found in certain number of generations.

If we compare our technique to the others used in fault tolerance domain, we achieved significant improvement. For example, a TMR will require 27 modules for 8×8 DCT computation and fault coverage would be limited to errors in only one for every three copies. On the other hand, the proposed method does not require additional modules during normal operation. It can handle even the case when 6 out of 8 modules are faulty. The test vector strategy would require 2^{96} vectors (8 values of 12 bit precision) to exercise all the logic inside a module computing a DCT function, which is practically impossible.

III. THE PROPOSED APPROACH

The reader is strongly suggested to see the hardware architecture shown in [8] which is not included here due to the limited space. Each PE in the DCT block is responsible for computing one coefficient of 8-point 1-D DCT.

A. Preliminary

An NMR (N-modular redundant) system is the one in which N copies compute the output and the effective output is the majority output. For N=2 (Dual Modular Redundancy), the discrepant output reveals that at least one of the two instances is faulty. Similarly, TMR (Triple Modular Redundancy) and higher can isolate faulty instances if the majority output is available. In this work, our assumption is that if at least one of two PEs is faulty in a pair under evaluation, they exhibit discrepancy at least once in a given *Evaluation Window* or their output remains the same. Similarly, no discrepancy between two module outputs exhibits their fault-free or *Healthy* nature.

The following is the terminology that we use in this paper. The terms *module* and *PE* are used interchangeably.

$PE_i.FS$ = Fitness state of i^th Processing Element
FS Enumeration = $\{Healthy, Suspect, Faulty\}$
$PE_i.function$ = The functionality assigned to a particular PE
RS = Reconfigurable Slack
Functional Modules = The modules providing normal throughput
$\{H\}$ = The set containing all the healthy modules from RS
$EvaluationWindow(W)$ = The duration through which the modules are evaluated for discrepancy check.

The cost of the fault isolation process is defined as:

$$cost = \sum_{i=1}^{I_{reqd}} I_{max,i}(W + \beta N_s) \tag{1}$$

Where I_{reqd} = Required number of iterations
$I_{max,i}$ = Number of times an i_{th} RS is reconfigured
β = Reconfiguration Penalty (PR time for one PE)
N_s=Number of Reconfigurable Slacks

B. *Fault Detection*

The fault detection methodology used here is observing peak signal to noise ratio (PSNR) of the recovered frames. Once this measure is below a given threshold, fault detection is asserted. In this way, we avoid all time exhaustive testing of the resources which will affect throughput. In addition, avoiding redundancy during normal operation is beneficial in terms of power and area requirement. Once fault is detected, the proposed fault isolation and recovery process are initiated.

C. *Fault Isolation and Recovery*

Algorithm 1 Fault Isolation and Recovery Process

```
Obtain current system parameters (e.g. Current DCT mode, Number of Blank PEs, $QP$ )
Determine the number of RS(s).
Apply the proposed fault isolation algorithm (Fig. 1)
Isolate faulty PEs (Bounded Number of Reconfigurations as in Fig. 5)
Reconfigure the DCT mode (full recovery or gracefully degraded mode)
```

Algorithm 1 describes the overall picture of the proposed fault isolation and recovery scheme, whereas the process is illustrated by flow chart in Fig. 1.Once fault occurs and every module needs to be examined, we proceed to identify the faulty PE(s) by employing *Reconfigurable Slack (RS)*. These slacks may be the blank PEs available in the system. The RS (or multiple slacks) is reconfigured with same functionality as that of the most important functional PE, for example, the module for computing DC coefficient. The location of faulty PE is detected by performing the discrepancy check in an NMR arrangement. For DMR, faulty status of one of the modules whereas for NMR faulty status of more than N-2 modules marks each of the instance as *Suspect*. Therefore, we proceed to reconfigure the RS with the 2^{nd} priority function and so on. Once an agreement between two modules in a complete evaluation window is observed, the two modules are declared as *Healthy* and their fitness state is updated. The identification of a healthy RS implies we do not need further slacks. A healthy RS can be used to check the fitness of all of the modules. The discrepancy of a suspected module in pair with a healthy module reveals its *Faulty* nature. On the other hand, an observed discrepancy between suspected modules does not provide any information and keeps them marked *Suspect*.

The fault isolation scheme is illustrated by an example in Fig. 2. Here normal operation is 8×8 DCT and therefore the PE1 to PE8 are providing the normal throughput for 1-D 8-point DCT. Two 1-D DCT operations are performed to compute 2-D DCT. PE9 is not being used by the system and reconfigured as blank. For this specific situation, assume that 2 out of total 9 PEs are faulty. The faulty PEs labelled 1 and 9 are shown by tagged boxes. We choose the blank PE as the *Reconfigurable Slack* (RS) for discrepancy check with PE1. Unfortunately we choose the RS which itself is faulty, yet we have no a-priori knowledge of its fitness state. Therefore, in the first iteration in which the RS is reconfigured 8 times for *Concurrent Error Detection* (CED) purpose, no information is obtained, every module's fitness is unknown and marked as *Suspect* (S). We proceede to the 2^{nd} iteration. The RS is moved from PE9 to PE8 and CED is performed with the functional modules, sequentially. A no discrepancy between PE8 and PE2 implies their *Healthy* (H) nature and therefore, PE1 and PE9 are marked as *Faulty* (F) as previously they did exhibit discrepancy with the *Healthy* module.

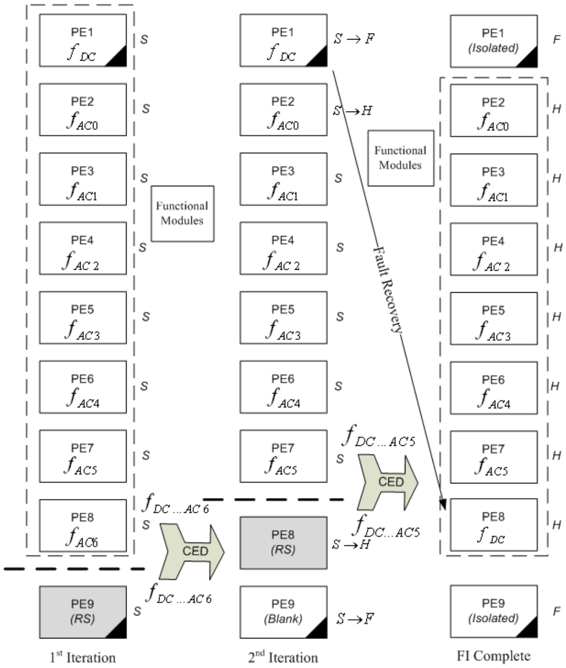


Figure 2. An Example of the Fault Isolation Scheme

Once the fault isolation process is complete and the healthy RS is identified, it can be used for error checking of all of the functional modules. If there are N healthy PEs available, fault recovery is made with no quality degradation at all. However, if there are fewer healthy modules than the current mode of DCT, then graceful degradation strategy is employed and only the important functions are retained.

IV. Simulation Results

Fault detection latency of the proposed approach is negligible, whereas latency of isolating faulty PEs is bounded

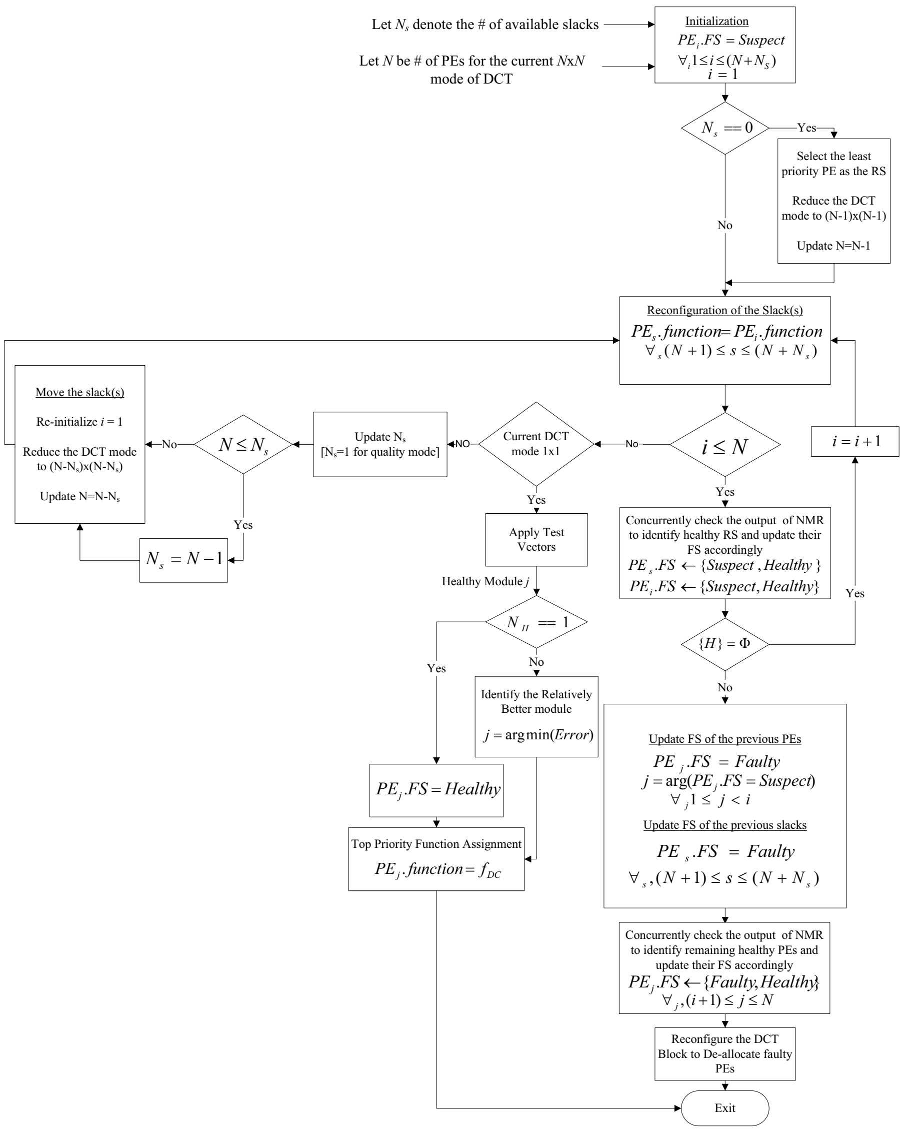

Let N_s denote the # of available slacks
Let N be # of PEs for the current NxN mode of DCT
Initialization
$PE_i.FS = Suspect$
$\forall_i 1 \leq i \leq (N+N_S)$
$i = 1$
$N_s == 0$
Yes
Select the least priority PE as the RS
Reduce the DCT mode to (N-1)x(N-1)
Update N=N-1
No
Reconfiguration of the Slack(s)
$PE_s.function = PE_i.function$
$\forall_s (N+1) \leq s \leq (N+N_s)$
Move the slack(s)
Re-initialize $i = 1$
Reduce the DCT mode to (N-N_s)x(N-N_s)
Update N=N-N_s
No
$N \leq N_s$
Update N_s
[N_s=1 for quality mode]
NO
Current DCT mode 1x1
No
$i \leq N$
$i = i+1$
Yes
$N_s = N-1$
Yes
Apply Test Vectors
Yes
Concurrently check the output of NMR to identify healthy RS and update their FS accordingly
$PE_s.FS \leftarrow \{Suspect, Healthy\}$
$PE_i.FS \leftarrow \{Suspect, Healthy\}$
Healthy Module j
Yes
$N_H == 1$
$\{H\} = \Phi$
Yes
No
Identify the Relatively Better module
$j = \arg\min(Error)$
No
Update FS of the previous PEs
$PE_j.FS = Faulty$
$j = \arg(PE_j.FS = Suspect)$
$\forall_j 1 \leq j < i$
Update FS of the previous slacks
$PE_s.FS = Faulty$
$\forall_s, (N+1) \leq s \leq (N+N_s)$
$PE_j.FS = Healthy$
Top Priority Function Assignment
$PE_j.function = f_{DC}$
Concurrently check the output of NMR to identify remaining healthy PEs and update their FS accordingly
$PE_j.FS \leftarrow \{Faulty, Healthy\}$
$\forall_j, (i+1) \leq j \leq N$
Reconfigure the DCT Block to De-allocate faulty PEs
Exit

Figure 1. The Fault Isolation and Recovery Process Flow Chart

as our algorithm follows a deterministic flow.

A. Latency of Fault Isolation

Fig. 3 shows the number of group reconfigurations required to isolate the faulty modules. For example, if the current DCT mode is 8 × 8 and one RS is available, approximately 50% of the fault scenarios are successfully isolated in the first iteration. As there are total 9 modules, there are 512 possible combinations of the faulty/healthy modules. Out of these 512 cases, about half of the cases require only one iteration to find a healthy RS. It is clear from the figure that increasing the number of RS, fault isolation process is speeded up. Fig. 4 shows the probability of isolating the faulty modules in the first iteration for different number of RS. As compared to the case when using one slack, more faulty situations are resolved using two slacks. Fig. 5 shows upper bound on the number of iterations when using different number of slacks for various fault situations. For all these figures, total number of PEs is 9 but can be extended without loss of generality. As an example, we require 2 iterations at most to isolate 7 faulty modules out of 9, when using 4 slacks.

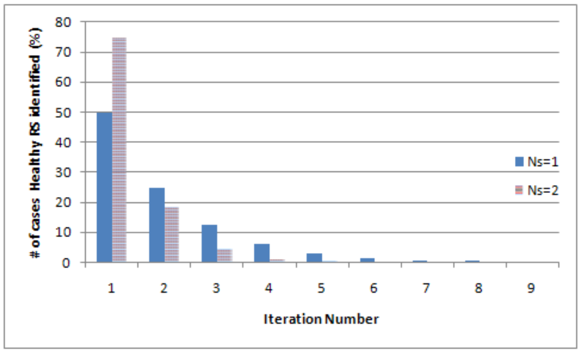


Figure 3. Number of Reconfigurations Required for Fault Isolation

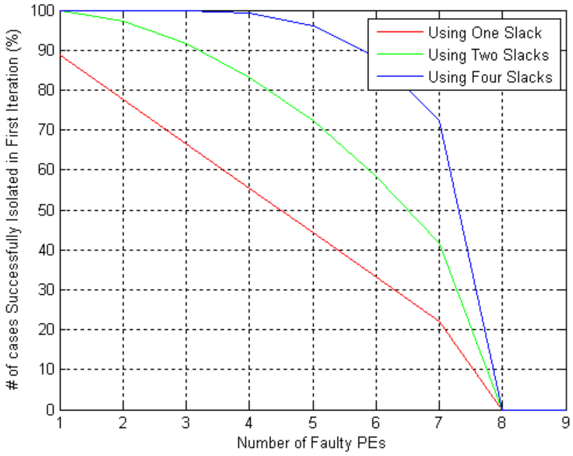


Figure 4. Probability of Finding Healthy RS in the First Iteration

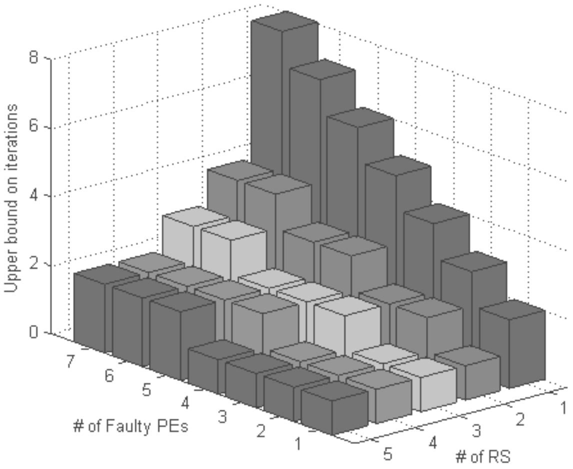


Figure 5. Upper Bound on Number of Iterations for Fault Isolation

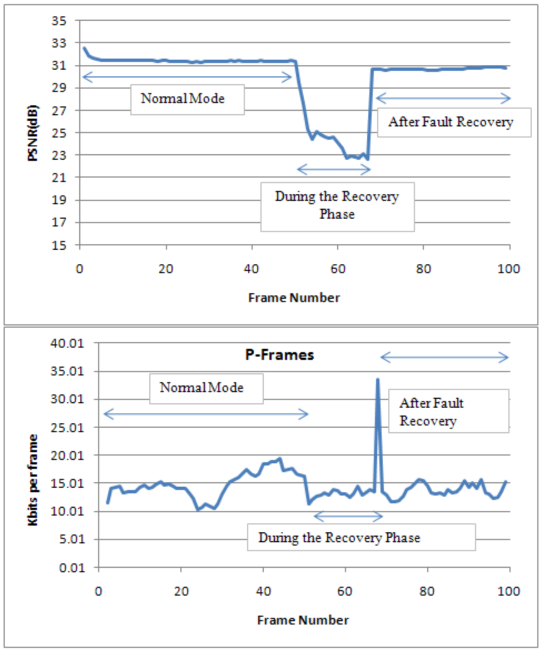


Figure 6. An Operational Example of the Video Encoder

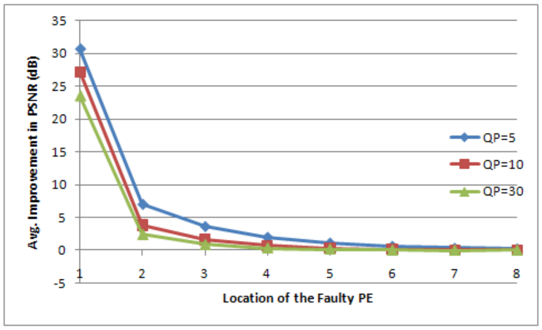


Figure 7. Improvement in Average PSNR after Fault Recovery

B. Performance Improvement

The proposed fault handling scheme is validated by observing the fault behavior of H.263 video encoder's DCT block. As shown in the Fig. 6, fault is injected in the DCT block at frame 50. The PSNR of the video sequence drops below threshold and the fault isolation scheme is called upon. After a few frames, the DCT block is recovered with only slight quality degradation. It may be noted, however, that we simulated a relatively worse scenario here where the number of healthy PEs available is less than N. Otherwise, there will be no quality degradation. Also, it is worthwhile observing that during fault isolation, considerable throughput is available. The bitrate for the fixed QP=10 is also shown in the 2^{nd} part of the Fig. 6.

The Fig. 7 shows improvement in PSNR in case of different faulty PEs location, for three QP values. The scheme is more advantageous for small QP value. Similarly, recovering from a case when a PE computing DC coefficient is faulty(i.e. PE1), is more beneficial.

The Fig. 8 shows PSNR results from the encoder with and without fault handling scheme. It is clear that a fault

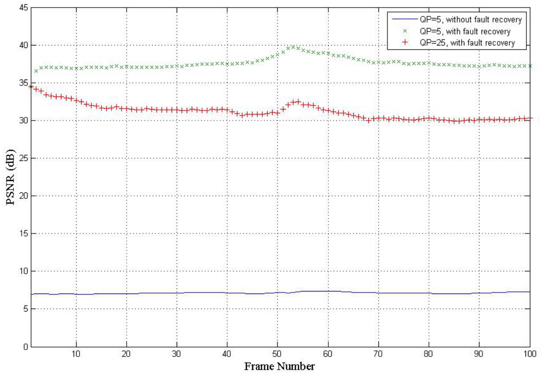
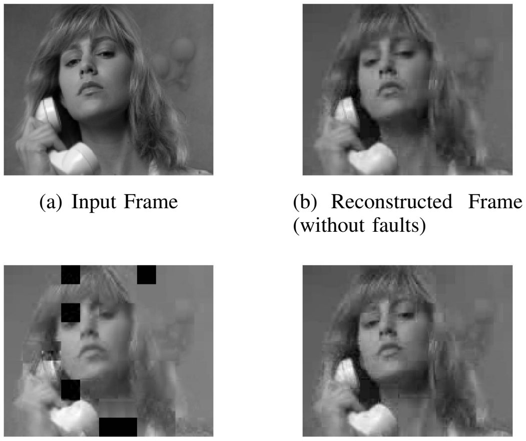


Figure 8. PSNR of Recovered Frames of a Video Sequence



(a) Input Frame

(b) Reconstructed Frame (without faults)

(c) The Effect of Faulty PE

(d) Recovered Frame (after fault-handling)

Figure 9. Qualitative Results of the Scheme (a), (b), (c), and (d)

Table II
IMPROVING THE COST SCORE OF FAULT RECOVERY USING INPUT SIGNAL CHARACTERISTICS

	QP=5	**QP=10**
Small Motion	301.7 msec	301.7 msec
Large Motion	1202.3 msec	301.7 msec

handling encoder even for a high QP value outperforms that with a low QP value without fault handling.

Fig. 9 illustrates the qualitative results of the approach. As it can be seen, the recovered image in Fig 9(d) after applying the fault handling scheme is visually much better than Fig. 9(c) where a faulty PE in the DCT block disrupts the image in the frame buffer.

In our work, the input signal characteristics are used in order to speed-up the fault isolation process. For example, depending upon the motion activity in a video sequence and the QP value, the required number of PEs used for DCT computation can be estimated. If the current video frames contain small motion activities, typically resulting in small residual errors after motion estimation, more PEs can be made free to be available to serve as the RS since DCT size can be reduced without much loss of visual quality. Table II lists the upper bound on fault isolation time for different types of video sequences. There are 9 total number of PEs used in this experiment, and one or 3 slacks are available depending upon the current mode of the DCT.

V. CONCLUSION

The experimental testing of the proposed algorithm shows promising results for fault handling operations. The number of iterations in fault isolation phase is bounded whereas the PSNR degradation is minimal throughout this duration by fully utilizing input signal characteristics to reduce DCT size. The priority of functions is taken into account to achieve the best possible solution in a fault scenario. We think that much broader applications will benefit from the proposed scheme of fault detection, isolation, and recovery on reconfigurable hardware fabric.

REFERENCES

[1] E. Stott, P. Sedcole, and P. Cheung, "Fault tolerant methods for reliability in FPGAs," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 8-10 2008, pp. 415 –420.

[2] M. Garvie and A. Thompson, "Scrubbing away transients and jiggling around the permanent: long survival of FPGA systems through evolutionary self-repair," in *On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International*, 2004, pp. 155–160.

[3] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial tmr," in *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, 2006, pp. 226 –232.

[4] A. Jara-Berrocal and A. Gordon-Ross, "VAPRES: a virtual architecture for partially reconfigurable embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 837–842.

[5] J. Emmert, C. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 216 –226, Feb. 2007.

[6] M. Gericota, G. Alves, M. Silva, and J. Ferreira, "Reliability and availability in reconfigurable computing: A basis for a common solution," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 11, pp. 1545 –1558, Nov. 2008.

[7] R. F. DeMara, K. Zhang, and C. A. Sharma, "Autonomic fault-handling and refurbishment using throughput-driven assessment," *Applied Soft Computing*, In Press, 25 January 2010.

[8] J. Huang and J. Lee, "Reconfigurable architecture for ZQDCT using computational complexity prediction and bitstream relocation," *Embedded Systems Letters, IEEE*, 2010.