Time-Multiplexed Online Checking: A Feasibility Study

Ming Gao, Hsiu-Ming (Sherman) Chang, Peter Lisherness, and Kwang-Ting (Tim) Cheng

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106, U.S.A.
{mgao, sherman, peter, timcheng}@ece.ucsb.edu

Abstract – **There is growing demand for online hardware checking capability to cope with increasing in-field failures resulting from variability and reliability problems. While many online checking schemes have been proposed, their area overhead remains too high for cost-sensitive applications. We introduce a time-multiplexed online checking scheme using embedded field-programmable blocks for checker implementation, which enables various system parts to be checked dynamically in-field in a time-multiplexed fashion. This incurs less area overhead, and could maintain fault coverage similar to traditional checkers. The test quality is studied using a probabilistic model. The implementation feasibility using a Field-Programmable Gate Array (FPGA) is demonstrated.**

Keywords – **Fault Detection, Online Testing, Dynamic Checker, Low-cost Checker, Availability**

1. Introduction

Semiconductor technology continues to progress toward greater speed and density, but designers have not managed to address the increasing variability and reliability of future nano-scale devices. Moreover, increasing design complexity as well as costly production testing and burn-in make it more difficult to ensure the shipment of failure-free chips. Additional in-field failure sources such as infant mortality, soft errors, and silicon aging contribute to chip failures as well.

To increase in-field chip availability, online checking followed by a fault circumvention process could be a promising direction. One possible scheme could monitor the chip using an online checker and, after error detection, enable self-repairing and failure recovery in the field. Such a solution would result in increased system availability and, in turn, a lower product return rate and service cost.

Although substantial efforts have been devoted to optimizing checking schemes and checker implementations for lower cost and greater fault coverage, the area overhead and performance penalties of existing approaches are still very significant. For combinational circuits, for example, a single parity bit checker – which works only for an odd number of bit errors – could incur an average area overhead of 77% and a performance penalty of 19% in an FPGA-implemented circuit for those benchmark circuits reported in [1]. The area overhead of an optimized online checker for a sequential circuit could exceed 100 percent [2]. Thus, these online checking solutions would not be suitable for cost-sensitive applications such as most consumer electronics.

To reduce hardware overhead, *roving emulation* -- a time-sharing method for offline fault detection -- was discussed in [3]. Probabilistic models were provided for estimating the fault detection latency of permanent faults using this offline method. No details regarding the hardware implementation of the method were reported. A concurrent fault detection method for combinational logic was described in [4], in which a roving type of checking scheme was mentioned for sharing the checker among identical Circuits Under Test (CUTs).

In this paper, we investigate an online checking scheme, *Time-Multiplexed Online Checking* (TMOC), which offers sufficient fault coverage with less overhead at the cost of increased *fault detection latency*. In TMOC, a design is partitioned into modules, each of which has its own TMOC checker. One or more embedded, field-reprogrammable blocks are used as shared checker spaces. Several TMOC checkers for different modules are sequentially and periodically mapped into a shared field-reprogrammable checker space in a time interleaved fashion. Utilizing this TMOC principle, different checker schemes, such as *duplex* checkers [5] or Error Detection Code (EDC) [6], can be chosen for different modules.

Current field-reprogrammable technology enables the time-interleaved implementation of TMOC checkers without disturbing normal system operation. The TMOC scheme can be applied to systems that can tolerate a certain level of *fault detection latency* and implemented either in FPGA or in SoC/SiP with embedded field-reprogrammable blocks.

In the next section, we propose an online checking scheme suitable for cost-sensitive designs that can be implemented in currently available fabrics. Section 3 presents detailed benefits and design trade-offs analyses. We propose a probabilistic model to estimate the *fault detection latency* and *probability of fault detection* of this scheme for permanent, transient, and intermittent faults. In Section 4, we show the implementation feasibility of the TMOC scheme. Solutions to the state synchronization problem for applying TMOC to sequential circuits are addressed. A case study based on applying TMOC to a JPEG encoder is presented in Section 5 to demonstrate the feasibility, benefits, and tradeoffs of employing TMOC in complex designs. Finally, we offer concluding remarks and discuss potential future work in Section 6.

2. Time-Multiplexed Online Checking

1081-7735/08 $25.00 © 2008 IEEE
DOI 10.1109/ATS.2008.23

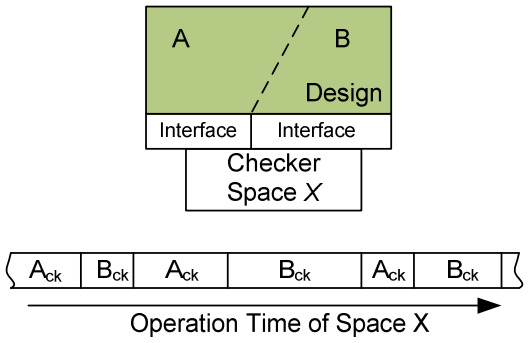

Figure 1: TMOC operating principles.

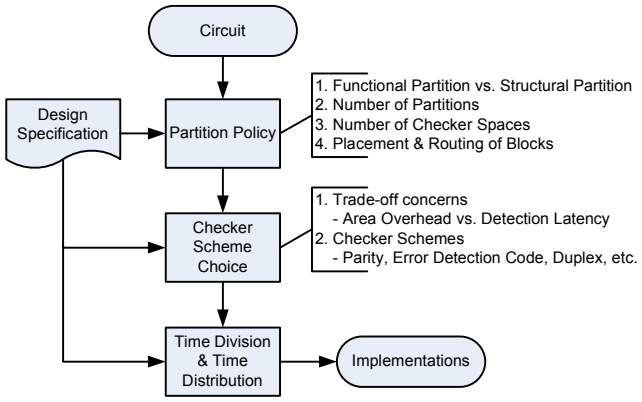

Figure 2: TMOC design flow.

2.1. Operating Principles

The TMOC scheme starts by partitioning a design into several modules, each of which has a TMOC checker. One or more field-programmable blocks, called *checker space(s)*, are used. Each checker is placed into a checker space in a time-interleaved fashion and each module is checked sequentially and periodically in a time-multiplexed fashion.

The TMOC scheme is illustrated in Figure 1. Here, the design is partitioned into two modules: *A* and *B*. Checkers A_{ck} and B_{ck} are the online checkers designated for modules *A* and *B*. The two checkers are time-multiplexed in the shared field-reprogrammable checker space *X*, Necessary interfaces isolate the signal and clock domains between the checker space *X* and the circuit. When checker space *X* is configured as checker A_{ck}, errors occurring in module *A* will be detected. Similarly, when this checker space *X* is configured as checker B_{ck}, it starts to monitor module *B*, and errors in module *B* could be detected. Checker space *X* will be periodically re-configured between checker A_{ck} and checker B_{ck} throughout the system operation, as shown in Figure 1. The design should operate without interruption even during re-configuration.

2.2. Design Parameters and Design Flow

Three primary design parameters – *partition policy, checker scheme,* and *time division and distribution* – should be considered when employing TMOC. A typical design flow shown in Figure 2, explores these design parameters.

Partition policy not only provides the information about the way a circuit is divided into modules, but also specifies the checker space's placement and use. Finer partitioning creates smaller module sizes and also increases the flexibility of the checker space assignment. The more modules that share the same time-multiplexed checker space, the less area overhead. However, having a large number of modules sharing a single checker space would require very complex routing and incur extra overhead for additional wiring. The number of partitions, the number and placement of checker spaces, and the mapping of modules to checker spaces must all be considered to effectively minimize area overhead and performance penalties.

The size of each module need not necessarily be the same. In general, it is beneficial to map similar sized checkers to the same space. This helps minimize area overhead since a shared checker space must be sufficiently large to accommodate the largest checker. The checker designated for each module need not be the same. Depending upon the fault coverage requirements and area overhead constraints, different checker schemes can be employed. Parity-based checking techniques [2] incur less implementation complexity and area overhead, but they also provide less fault coverage than the EDC-based checking techniques [6]. While incurring less area overhead than *diverse duplex* checking techniques [5], *identical duplex* checking techniques [7] are vulnerable to Common Mode Failures (CMF) [8]. Without loss of generality, we employ identical duplex checking as an exemplar checking scheme in this paper. The proposed method is not tied to any specific checking scheme and other checking schemes can be employed interchangeably.

While the previous two parameters are determined during the design phase, *time division* and *distribution* are adjustable in the field. Thus, the *fault detection latency* can be dynamically adjusted even after the placement and routing of system. Here, *time division* refers to the minimum granularity of each user-defined time slot in which a checker continues monitoring a module. *Time distribution* is the time slot assignment policy among modules.

These specific design parameter choices are design - and implementation-dependent. In Section 3, we illustrate how the *number of partitions* and the *test window* of each partition affect the *test quality*. A complete analysis involving all parameters and existing design constraints is beyond the scope of this paper and will require further research.

3. Analysis

3.1. Area Overhead

Consider a circuit that is partitioned into *N* modules of equal size and the size of the checker for each module – including the interfaces, extra routing and control circuits – is β times the size of the module. In practice, β is often greater than one, especially for duplex checkers. Let *U* denote the size of the original design. Thus, U/N is the size of each module and $\beta \times U/N$ is the area overhead incurred by each

checker. If these modules share M checker spaces and each checker space is used to check the same number of modules, the area overhead AO would be:

$$AO = \frac{M \times \beta U}{N} \quad (1)$$

Equation (1) indicates that given a fixed number of checker spaces, more partitions yield less area overhead as more checkers share a checker space; however, when the ratio of checker spaces to partitions, M/N, decreases, the β factor will increase. β cannot be accurately estimated since it is platform and design dependent.

Assuming all checker spaces and circuit partitions are the same sized, Equation (2) shows a first-order estimation of AO, in which S_{if} denotes the size of interfaces and extra routing circuitry for each checker space, and S_{ck} is the size of each checker space.

$$AO = N * S_{if} + M * S_{ck} \quad (2)$$

3.2. Test Quality

In this section, we propose a probabilistic model to estimate the test quality achieved by TMOC for three classes of faults -- permanent, transient and intermittent. Our analysis offers the following unique aspects:

1. Besides permanent faults [3], our analysis addresses the detection of transient faults and randomly repeatable intermittent faults.
2. In contrast to the previous intermittent fault detection models [4][9], we model an intermittent fault as a series of independent transient faults separated by random intervals. Two independent random variables -- *Active-Period (T_{AP})* and *Time-Between-Active-Periods (T_{BA})* -- are defined to characterize the intermittent faults.
3. To quantify the test quality of a checker design employing TMOC, we examine two primary metrics, *probability of fault detection* and *fault detection latency*.

3.2.1. Parameters

1. TMOC design parameters

N: The number of design modules sharing one checker space. $N = 1$ means a dedicated checker is employed for each module.

T_{TW}: The time duration of one test window for each module. Assuming a round-robin checking scheme is implemented among the N modules, the test window for a given module will recur with a period of $N*T_{TW}$.

T_{OV}: Time overhead incurred due to checker reconfiguration and synchronization.

m: The number of clock cycles that the module-under-checking is really been checked during one test window. Note that m is proportional to (T_{TW} - T_{OV}).

2. Random variables for fault characteristics

We model an intermittent fault as a series of independent transient faults, each of which is characterized by its *active period* and successive occurrences are separated by a random interval.

$T_{AP}(i)$: The time duration of the i^{th} *active period* of an intermittent fault. Note that transient faults are a special case of intermittent faults with only one *active period*. Thus, T_{AP} is used in place of $T_{AP}(i)$. Similarly, permanent faults are a special case for which the active period T_{AP} is infinity.

$T_{BA}(i)$: The period between the i^{th} and $(i+1)^{th}$ *active periods* of an intermittent fault.

3. Detectability of input patterns

q: The probability that an input vector detects the fault during on-line checking, given that the module is faulty.

The outputs of our fault detection analysis are:

DL: Fault detection latency which is defined as the time interval between the occurrence of a fault and its detection.

PD: Fault detection probability which is defined as the probability of detecting a fault before the fault vanishes, given that the fault exists in the system.

3.2.2. Probabilistic Model

In the following analysis, we focus on quantifying TMOC with respect to the increase in *fault detection latency* (*DL*) and the *probability of missing a fault* (1-*PD*), in comparison with a dedicated checker.

During a given T_{AP}, the average number of clock cycles when a module is indeed checked can be expressed as:

$$k = \frac{T_{AP}}{N} * \frac{m}{T_{TW}}, \quad (3)$$

Assuming the fault is detected during this T_{AP}, the average clock cycles required for fault detection would be:

$$r = \frac{\sum_{n=1}^{k} n * q(1-q)^{n-1}}{\sum_{n=1}^{k} q(1-q)^{n-1}}, \quad (4)$$

Based on these parameters, the detection latency can be expressed as the *average time to detect a fault (D)* plus the *average waiting time before checking begins (W)*:

$$DL = D + W = \frac{r}{m} * N * T_{TW} + \frac{1}{2} * (N-1) * T_{TW} \quad (5)$$

In the expression of D, the term r/m can be interpreted as the average number of the test windows required for fault detection, and the term $N*T_{TW}$ is the interval between two successive test windows for checking a module. In the expression of W, $(N-1)*T_{TW}$ is the worst case initial waiting time for a module. The initial waiting time is the interval between the occurrence of a fault and the beginning of the following T_{TW}.

We made the following assumptions in our analysis. (a) There are only single-faults. No multiple faults occur simultaneously in the circuit. (b) $T_{AP}(i)$*'s* are independent

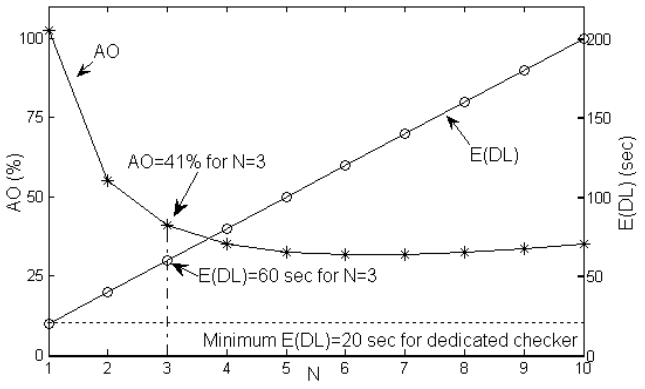

Figure 3: *AO* and E(*DL*) versus *N*. Parameters: *M*=1, S_{if}=25, S_{ck}=1000, q=10^{-7}, T_{TW}=20 ms, T_{OV}=10 ms.

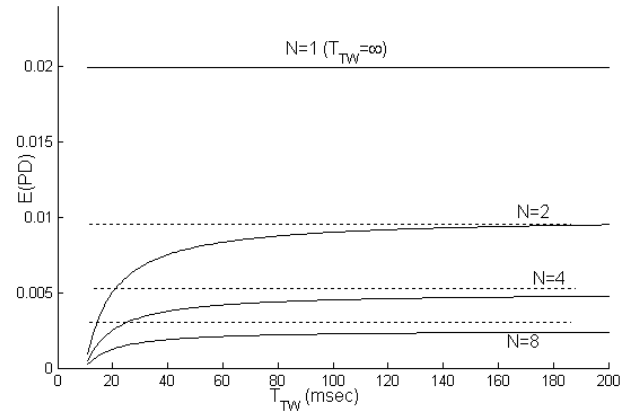

Figure 4: E(*PD*) versus T_{TW} (Transient Faults). Parameters: q =10^{-5}, T_{AP}=2 ms, T_{OV}=10 ms.

and identically-distributed (i.i.d.) random variables. We denote its expected value as E($T_{AP}(i)$)=C_{AP}. Similarly, $T_{BA}(i)$'s are i.i.d. and the expected value E($T_{BA}(i)$) is denoted as C_{BA}. Also, $T_{AP}(i)$ and $T_{BA}(j)$ are independent, for any pair of i and j. (c) Round-robin scheduling is used as the time interleaving policy for the TMOC checker space.

For permanent faults, T_{AP} = ∞, T_{BA} = 0, and k = ∞. Therefore, we can conclude that:

$$DL = \left(\frac{1}{qm}+\frac{1}{2}\right) * N * T_{TW} - \frac{T_{TW}}{2} \tag{6}$$

$$PD = \lim_{k\to\infty} \sum_{n=1}^{k} q(1-q)^{n-1} = 1 \tag{7}$$

For transient faults, T_{AP} < ∞ and T_{BA} = ∞. Let $f_{ap}(x)$ denote the probability density function of T_{AP}, i.e., $f_{ap}(x) = P\{T_{AP} = x\}$. Note that the expected value of *DL* is derived under the condition that such a transient fault will be detected before it vanishes.

$$E(DL) = \sum_{x=1}^{\infty}\left[\left(\left(\frac{r}{m}+\frac{1}{2}\right) * N * T_{TW} - \frac{T_{TW}}{2}\right) * f_{ap}(x)\right] \tag{8}$$

$$E(PD) = \sum_{x=1}^{\infty}\left[\left(1-(1-q)^{k}\right) * f_{ap}(x)\right] \tag{9}$$

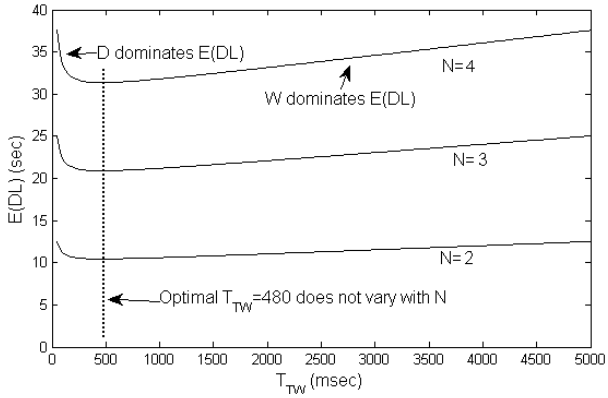

Figure 5: E(*DL*) versus T_{TW} with different *N*. Parameters: q=10^{-7}, C_{AP}=5 ms, C_{BA}=400 ms, T_{OV}=10 ms.

For intermittent faults, T_{AP} < ∞; T_{BA} < ∞. Let DL_i denote the fault detection latency of the i[th] active period of an intermittent fault. Since DL_i's have the same mean value (from assumption (b) mentioned earlier), we denote E(DL_1) = E(DL_2) = E(DL_3) = …=C_{DL}. Similarly, let PD_i denote the probability that the fault detection occurs during the i[th] active period of an intermittent fault. We denote E(PD_1) = E(PD_2) = …=C_{PD}. The total expected *PD* and *DL* for an intermittent fault are:

$$E(PD) = \sum_{n=1}^{\infty}\left[C_{PD} * (1-C_{PD})^{n-1}\right] = 1 \tag{10}$$

$$E(DL) = \sum_{n=1}^{\infty}\left\{C_{PD} * (1-C_{PD})^{n-1} * \left[(C_{AP}+C_{BA}) * (n-1) + C_{DL}\right]\right\} \tag{11}$$

3.2.3. Parameter Sensitivity Analysis

By varying the values of various design and fault parameters defined earlier, we could evaluate the impact of these parameters on test quality.

Figure 3 shows the area overhead (*AO* from Equation 2) and the average *DL* for a permanent fault as functions of *N*, assuming all the partitions share one checker space (i.e. *M* = 1). When *N* is small, increasing *N* reduces the size of the checker space. However, when *N* becomes sufficiently large, the checker interfaces and required extra routing circuitry will offset the checker size reduction benefit. This causes *AO* to level out or even increase. The effect of *N* on *DL* is more direct: *DL* increases linearly as *N* increases. These two trends can be used to determine a suitable value of *N*. As an example, *N=3* in this figure shows a design striking the best balance for meeting both *area overhead* and *fault detection latency* constraints from the design specification.

In addition to increasing *DL*, the curves in Figure 4 show that increasing *N* reduces the *PD* of transient faults. As T_{TW} approaches infinity, the *PD* approaches a constant as shown in Equation (12). T_{TW} is always infinite with a dedicated checker (*N*=1). The fault detection probability is limited by the probability of a fault being activated and observed.

$$\lim_{T_{TW}\to\infty} PD = \lim_{T_{TW}\to\infty}\left(1-(1-q)^{\frac{T_{AP}}{N} * \frac{m}{T_{TW}}}\right) = 1-(1-q)^{\frac{T_{AP}}{N}} \tag{12}$$

In Figure 5, these curves for intermittent faults can help find a proper T_{TW} to meet a certain *DL* constraint. The "elbow" of each curve indicates a theoretically optimal T_{TW} for minimizing the *DL* for a given fault. When the T_{TW} is small and dominated by T_{OV}, the *average time to detect a fault (D)* dominates the *DL*. This makes the *DL* significantly increase as the T_{TW} decreases. However, as the T_{TW} increases beyond the optimal point, few test windows are required to detect a fault and the *average waiting time before checking begins (W)* will dominate the *DL*.

4. TMOC Implementation Feasibility

4.1. TMOC Implementation Platforms

The proposed TMOC scheme works for designs that are capable of dynamic in-field reconfiguration –a SoC/SiP with embedded field-reconfigurable blocks, an FPGA-based design, or a heterogeneous multi-core system with spare or idle cores.

It becomes increasingly common to embed reprogrammable blocks in a SoC design, or to embed an FPGA die in an SiP. Examples include Triscend's E5, IBM's Cu08 family, and Atmel's FPSLIC. It has been illustrated that such solutions can efficiently improve the quality of manufacturing test [10], post-silicon validation [11], and silicon debug [11]. Furthermore, after shipment, these blocks can function as in-field reconfigurable blocks for other purposes. By utilizing them as the checker spaces, we illustrate that the TMOC scheme can perform in-field error detection without incurring much extra hardware overhead. Similarly, for multi-core SoCs, some cores can be used as spares to improve the die yield [12]. Such spare cores, along with the idle functional cores, could be reused to perform in-field TMOC. Although our current experiments and the case study were implemented on a pure FPGA platform, the TMOC scheme could be employed for SoC/SiP designs with embedded FPGAs and for both homogeneous and heterogeneous mult-core chips.

For FPGA-based designs, Xilinx®'s Virtex series FPGA, for example, supports an in-field reprogrammable feature called *Partial Reconfiguration* (PR) [13]. In PR, portions of an FPGA can be re-reprogrammed without interrupting the functional operation of the rest of the FPGA. When a TMOC scheme is implemented on an FPGA, the checker spaces are defined as the *dynamic regions* [13] that can be in-field reconfigured with pre-computed bit-streams. These checker spaces are separated from the circuit modules with a specific tri-state buffer called the *bus macro* and the inputs to a circuit are fed into both the module-under-check and the checker space. We have successfully demonstrated the TMOC scheme on a set of arithmetic circuits on a Virtex II Pro/ML321 board.

4.2. Solutions to the State Synchronization Problem

State synchronization between a sequential design module F and its checker F_{ck} should be considered during the reconfiguration of the TMOC checker spaces.

A means of synchronizing F and F_{ck} is to directly copy the value of the state registers from F to F_{ck} if the initial values are different. Such a mechanism is called *state-copy*. In our FPGA experiments, this worked well for some simple FSM designs.

We propose a *state-flush* mechanism as another method to synchronize F and F_{ck}. In many applications, two successive data units are state-independent. The synchronization could be achieved automatically when input data stream into both design module and its checker until they both reach the same state. For example, in the following case study using JPEG encoder, the processing unit is an eight-by-eight pixel matrix, and two successive matrices are state-independent. A counter was used to guarantee the synchronization by simply counting input pixels and detecting whether enough matrices have flushed the pipeline of checker.

Generally speaking, the *state-copy* mechanism might be costly for complex designs due to the wiring and routing issues. Especially, for an encrypted commercial *Intellectual Property* (IP) core whose implementation information is not available, employing the *state-flush* mechanism would be a good choice. However, this would incur some extra time overhead - the result of flushing state registers

5. Case Study: JPEG Encoder

We apply TMOC to a modified open-source JPEG encoder [14] to further evaluate the area overhead and the fault detection latency in realistic designs. The implementations were on a Xilinx Virtex II Pro FPGA on a BEE2 system [15].

Following the TMOC design flow in Figure 2, we first functionally partition the design into five modules: *Color Conversion (CC)*, *Discrete Cosine Transform (DCT)*, *Quantization (QT)*, *Fist-In-First-Out (FIFO)*, and *Huffman Encoding (HE)*. A *duplex* checker was built for each module, and the *state-flush* mechanism was employed. In implementations J2 and J3, checker space CK1 was dedicated to checking the DCT module. In J1, those small partitions were grouped into one module for more balanced partitioning.

The important design metrics of each TMOC configuration are summarized in Table 1. The third column shows the area overhead caused by the checker circuits. The fourth column shows the fault detection latency estimated using the analytical model proposed in Section 3, with the following assumptions ("w/o TMOC" means N = 1 and a dedicated checker is employed.): (a) The detection probability q for intermittent fault is 10^{-7}, and T_{AP} and T_{BA} are geometrically distributed with their mean values that C_{AP} = 10ms, and C_{BA} = 50ms; (b) T_{TW} = 100ms, and T_{OV} =5ms; (c) The system clock frequency is 100MHz. The fifth column shows, in percentage, the increase in fault detection latency compared to a dedicated checker.

As seen in Table 1, when we applied a proper partitioning policy, CK2 of J3 for instance, two duplex checkers sharing one checker space incurred 54.17% area overhead -- less than CED-based *duplex* checkers, which typically required more than 110% area overhead [1][2]. However, the area overhead

Table 1: Implementation Results of TMOC Using JPEG Encoder

	Number of modules sharing one checker	Area Over--head	*DL* (ms) (w/o) / (w/) TMOC	Increase of *DL*
J1	CK1: 2	67.55%	654/1333	103.8%
J2	CK1: 1	100%	654/654	0%
	CK2: 4	81%	654/2691	311.5%
J3	CK1: 1	100%	654/654	0%
	CK2: 2	54.17%	654/1333	103.8%
	CK3: 2	97.56%	654/1333	103.8%

of some non-dedicated checkers in Table 1 is still close to 100%, because the DCT module (1650 slices) dominates the overall area of the JPEG encoder (2376 slices). The DCT IP core cannot easily be further functionally partitioned. This is common in designs with reused IP cores. For such cases, structural partitioning should be employed for finer design partitions granularity to help reduce area overhead. In J2, for example, this technique can be applied as follows:

Step 1: Partition the DCT module (1650 slices) into four modules. Each is smaller than the HE module (520 slices);

Step 2: Combine CC (66 slices), QT (78 slices), and FIFO (13 slices) into a group G1 (66+78+13 = 157 slices);

Step 3: HE, G1, and the four modules from DCT share one checker space CK1.

With this implementation, the overall *area overhead* can be reduced to around *21.89%* with an increase in the *fault detection latency* by *519%*，from 0.654 sec to 4.049 sec, for the same intermittent fault.

6. Conclusion

We proposed a *Time-Multiplexed Online Checking* scheme that could achieve fault coverage similar to that of conventional online checking methods. Our method offers significantly less area overhead at the cost of some increased fault detection latency. Such an advantage makes TMOC a prominent candidate for implementing online checking in cost-sensitive electronics. To demonstrate the feasibility of TMOC, we used field-reprogrammable blocks to implement the online checkers and to periodically check the pre-partitioned design modules. This online checking scheme can capture all permanent and repeatable intermittent errors in a circuit, but it may miss some transient or soft errors that occur when the module is not monitored and that do not occur again later when the module is monitored. The probability of having such non-repeatable transient errors should be low, but further validation is needed to justify this claim.

Our on-going research efforts include applying the proposed online checking scheme to a modern complex design. Using such a design as a driver, we will perform further investigation regarding the physical implementation constraints of TMOC, such as power consumptions, partitioning policy, and performance penalty.

Acknowledgements

The authors acknowledge the valuable suggestions from Professor Jin-Fu Li at the National Central University, Taiwan (R.O.C), and Professor Ryan Kastner at the University of California, San Diego, U.S.A.. The authors also acknowledge the support of the Gigascale Systems Research Center (GSRC), one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program as well as GSRC BEE2 initiative program.

References

[1] M. K. Stojcev, G. L. Djordjevic, and T. R. Stankovic, "Implementation of Self-checking Two-level Combinational Logic on FPGA and CPLD Circuits," *Microelectronics Reliability*, vol. 44, pp. 173-178, Jan. 2004.

[2] C. Zeng, N. Saxena, and E. J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection," *Proceedings of the 1999 IEEE International Test Conference*, pp. 672-679, Oct. 1999.

[3] M. .A. Breuer and A. A. Ismaeel, "Roving emulation as a fault detection mechanism," *IEEE Trans. Comput*. C-35 (11): 933-939, 1986.

[4] R. Sharma and K. K. Saluja, "An implementation and analysis of a concurrent built-in self-test technique," in *Fault Tolerant Computing Symposium*, 1988, pp. 164-169.

[5] S. Mitra and E. J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?," *Proceedings of 2000 IEEE International Test Conference*, pp.985-994, Oct. 2000.

[6] D. Das and N. A. Touba, "Synthesis of Circuits with Low-Cost Concurrent Error Detection based on Bose-Lin codes," *Proc. IEEE VLSI Test Symp.*, pp. 309-315, 1998.

[7] E. J. McCluskey, "Design Techniques for Testable Embedded Error Checkers," *IEEE Computer*, vol. 23, no. 7, pp. 84-88, July 1990.

[8] J. H. Lala and R. E. Harper, "Architectural principles for safety-critical real-time applications," Proc. of the *IEEE*, vol. 82, pp. 25-40, 1994.

[9] A. A. Ismaeel and R. Bhamagar, "Test for detection & location of intermittent faults in combinational Circuits," *IEEE Trans. Reliability*, Vo1.46, No.2, pp.269-274, 1997.

[10] G Zeng and H Ito, "Hybrid BIST for system-on-a-chip using an embedded FPGA core," Proceedings of 22nd *IEEE VLSI Test Symposium*, pp. 353-358 April 2004.

[11] DAFCA Inc., "In-Silicon Solutions for Silicon Debug," whitepaper, http://www.dafca.com/literature/whitepapers.php

[12] Electronic News, "Could 10-20% yields for Cell processors lead to problems for Sony PS3?" interview of Tom Reeves, http://www.edn.com/article/CA6350202.html, July 7, 2006

[13] Xilinx Inc., "XAPP290: Two Flows for Partial Reconfiguration: Module Based or Difference Based," v. 1.2, Sept. 2004.

[14] V. Lorenzo, "JPEG Hardware Compressor," OpenCores.org Website, http://www.opencores.co.uk/projects.cgi/web/jpeg/, July 2005.

[15] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: A High-end Reconfigurable Computing System," *IEEE Design and Test of Computers*, vol. 22, no.2, pp. 114-125, Mar.-Apr. 2005.