電気学会論文誌 D（産業応用部門誌）
IEEJ Transactions on Industry Applications
Vol.141 No.2 pp.93–99　DOI: 10.1541/ieejias.141.93

論　文

動的部分再構成を用いた耐故障化手法の Xilinx Zynq-7000 SoC による試作

非会員　荻堂　盛也*　正　員　市川　周一*a)　非会員　藤枝　直輝**
上級会員　山田　親稔***　非会員　宮城　桂***

Experimental Implementation of Fault Tolerance Using Dynamic Partial Reconfiguration on Xilinx Zynq-7000 SoC

Seiya Ogido*, Non-member, Shuichi Ichikawa*a), Member, Naoki Fujieda**, Non-member,
Chikatoshi Yamada***, Senior Member, Kei Miyagi***, Non-member

（2020年3月6日受付，2020年5月29日再受付）

In our previous paper, the authors proposed a fault tolerant system that adopts field programmable gate arrays (FPGA) with dynamic partial reconfiguration (DPR), based on autonomous control of reconfiguration. This study presents an experimental implementation of the proposed system that utilizes the DPR feature of Xilinx Zynq-7000 SoC. The control logic of DPR is implemented as a Linux software on the embedded ARM processor of Zynq-7000. DPR is invoked via PCAP, which is the dedicated interface for the embedded ARM processor. Four tiles (reconfigurable areas) are prepared and dynamically reconfigured to avoid the firm error of SRAM-type FPGAs. An experimental fault-tolerant system with triple redundancy and logic roving is implemented, and the measurement results of the reconfiguration time and data transfer time are presented.

キーワード：動的部分再構成（DPR），FPGA，組み込みシステム，組み込み Linux
Keywords: Dynamic Partial Reconfiguration (DPR), FPGA, embedded systems, embedded Linux

1. はじめに

宇宙産業など高い信頼性が求められる状況下では，面積冗長化手法が多く採用される[1][2]。面積冗長化では，等価な回路を複数実装することで信頼性を保証しているため，大きな面積オーバーヘッドが発生する。そこで近年，FPGA（Field Programmable Gate Array）などの再構成型デバイスを用いた再構成型耐故障アーキテクチャが注目されている。再構成型耐故障アーキテクチャでは，故障が発生すると故障箇所と同等の回路を未使用領域に書き込み，故障状態から復帰する。このため，常に等価な回路を複数実装する面積冗長化に対して，高い面積効率が得られる。

著者ら[3]は，動的部分再構成（DPR；dynamic partial reconfiguration）[4]を用いた耐故障化手法の提案と，その自律的な制御方法について検討を行ってきた。この提案手法[3]では，耐故障動作を司る RM（Recovery Manager）を FPGA 上に配置し，定期的に再構成することで信頼性を保証しようとした。しかし RM の機能が増えると実装が複雑になり，既存の FPGA デバイスで実装すると大きなオーバヘッドを伴う。そこで本研究では，応用研究を進展させるための評価基盤として，RM の機能を組込みプロセッサ上のソフトウェアとして実現した。

近年の FPGA デバイスでは，組込みシステム実現のためにプロセッサを搭載する製品が増えている。再構成論理を利用してマイクロプロセッサを構成する場合（ソフトコアプロセッサ）と，再構成論理部分と別にマイクロプロセッサを

a) Correspondence to: Shuichi Ichikawa. E-mail: ichikawa@ieee.org
* 豊橋技術科学大学 電気・電子情報工学専攻
〒441-8580　愛知県豊橋市天伯町雲雀ケ丘 1-1
Department of Electrical and Electronic Information Engineering, Toyohashi University of Technology
1-1, Hibarigaoka, Tempaku-cho, Toyohashi, Aichi 441-8580, Japan
** 愛知工業大学工学部 電気学科
〒470-0392　愛知県豊田市八草町八千草 1247
Faculty of Engineering, Aichi Institute of Technology
1247, Yachigusa, Yakusa-cho, Toyota, Aichi 470-0392, Japan
*** 国立高等専門学校機構 沖縄工業高等専門学校
〒905-2192　沖縄県名護市字辺野古 905 番地
National Institute of Technology, Okinawa College
905, Henoko, Nago, Okinawa 905-2192, Japan

搭載する場合（ハードコアプロセッサ）に分けられる。本研究では，実装プラットホームとして Xilinx 社の Zynq-7000 SoC [5] を用いた。Zynq-7000 SoC は，ARM アーキテクチャのハードコアプロセッサ（PS；processing system）と FPGA 部（PL；programmable logic）を混載した SoC（System on Chip）である。

RM を PS 上のソフトウェアとして実装すると，RM が単一故障点となり信頼性を損なう可能性がある。しかし既存の高信頼性手法により，PS 部分の信頼性を維持できる。例えば，2 つのプロセッサを用いて OS から二重系として動作させることが考えられる（Zynq にはハードコアを複数搭載するモデルがある）。メモリエラーについては ECC（Error Correction Code）を利用する，チェックポイント・リスタートを実装する，などの手法が知られている。

本研究ではハードコアプロセッサを利用したが，将来的にはソフトコアプロセッサを利用して故障時に DPR で RM を復旧する等の検討も可能である。しかしそれは本研究の次のステップであり，本研究では触れない。本研究の目的は再構成型耐故障アーキテクチャの実装であるが，この実装手法は広範囲に適用可能である。例えば，伝統的な多重化による耐故障化も実装可能であるし，ソフトウェアの一部をハードウェア化して知的財産を保護する研究 [6][7] にも利用できる。その意味でも有効性の高い実装手法であるといえる。

以下，2 章で研究背景と技術要素を説明し，次に 3 章で本研究における実装方法を述べる。4 章では，応用例と動作確認結果を報告する。なお本論文は電気学会次世代産業システム研究会で発表した原稿 [8] に加筆修正を施したものである。

2. 背　景

〈2･1〉 FPGA における故障　FPGA における故障は，ソフトエラー，ファームエラー，ハードエラーの 3 種類に分類される。

ソフトエラーは，荷電粒子が半導体基板に衝突する際に発生するもので，記憶素子（FF やラッチ）の値の反転を引き起こす。ソフトエラーは過渡故障に分類され，次のクロックで新たな値が書き込まれれば以後正しく動作する。

近年の FPGA のほとんどは SRAM 型であり，論理構成が SRAM（構成メモリ）に書き込まれている。構成メモリのセルがソフトエラーを起こすと，論理回路として異常（故障）を起こすことがある。この故障はファームエラーと呼ばれており，構成データを再度書き込むまでは直らない。構成データを書き直すと以後正しく動作するので，故障部分の DPR により復旧することができる。

素子の経年劣化や，荷電粒子によるラッチアップなどで，素子自体が不可逆に故障することもある。こうした故障をハードエラーという。再構成型耐故障アーキテクチャでは，故障部分の機能を論理の未使用部分に構成することにより，ハードエラーから復旧することができる。

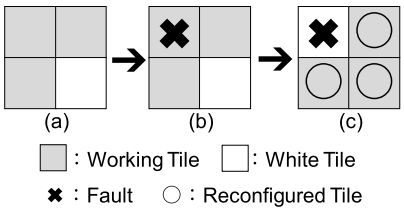

Fig. 1.　Concept of tile fault tolerant system [3].

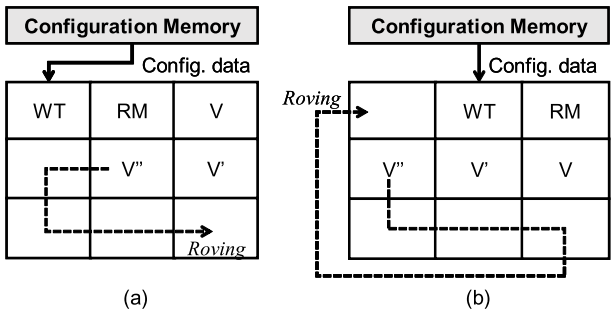

Fig. 2.　Fault-tolerant methods proposed by the authors [3].

〈2･2〉 タイルフォールトトレラント　まず，再構成型耐故障アーキテクチャの例として，タイルフォールトトレラント [9][10] について説明する。この手法では，FPGA 内をタイルと呼ばれるユニットに分割して用いる。FPGA 上に作成される回路はタイルの集合体として表現され，タイルに故障が発生した場合は余白のタイルを故障箇所と同等に再構成することで復旧する。

Fig. 1 は，再構成により故障状態から回復するようすを表したものである。故障前の正常状態（Fig. 1(a)）から，左上のタイルが故障したとする（Fig. 1(b)）。故障したタイルを避けて，残り 3 つのタイルで機能を実現することにより，機能を回復する（Fig. 1(c)）。

この手法は多重化と比べて，故障回数あたりの面積効率に優れているのが特徴である。ただし再構成の度にタイル同士をつなぐ配線が複雑になって，配線遅延による処理能力低下が問題となる。

〈2･3〉 著者らが過去に提案した耐故障化手法　次に，著者らが過去に提案した耐故障化手法 [3] を説明する。その概略を Fig. 2 に示す。

本手法では，FPGA 内の利用可能領域をタイル状に分割して用いる。Fig. 2 において，WT（Working Tile）は目的の回路機能，RM（Recovery Manager）は耐故障動作の制御回路，V（Voting Circuit）はタイルの異常を検出するための多数決回路を表す。構成メモリ（Configuration Memory）には，各タイルの論理構成データが格納されている。ファームエラー回避のため，構成データは一定時間ごとに書き換える。その際，各タイルの機能を未使用タイルに巡回移動することにより，機能を維持しながら書き換えを行う（Roving）。未使用タイルが故障していないことを確認するため，テスト用回路を 3 つのパネルに書き込んで，一

致を確認する（V，V’，V”）。

本手法では機能を各タイル毎に分割実装するため，設計の制約は強くなるが，再構成に伴う実装上の問題は軽減される。問題点として，Xilinx 社の DPR では各タイルに実装するすべての回路の構成データを用意する必要があるため，大きな構成メモリが必要になる（タイル数 × 機能数）。また RM 自体を移動させるため設計が複雑になる。

〈2･4〉 動的部分再構成 FPGA は通常，論理機能を書き換える際，回路全体を停止させたうえで再構成および再配線を行う。しかし，Xilinx 社の DPR [11][12]では，論理動作を停止させずに一部の論理機能を再構成することができる。

まず FPGA の領域をパーティションで区切り，その中に DPR 可能な領域（DPR 領域）を作成する。Xilinx 社の FPGA では，構成データの入ったファイルをビットファイルと呼ぶ（拡張子が bit であるため）。FPGA 全体をフルビットファイルで構成した後，DPR 領域の回路情報を持つパーシャルビットファイルを読み込むことにより，他の領域の回路動作に影響を与えずに部分的な再構成を行うことができる。回路を除去するには，ブランキングビットストリーム [11][12]を書き込む。

PCAP（Processor Configuration Access Port）は，Zynq-7000 等の SoC 型 FPGA を構成する主な手段である [11][13]。PS 部のプロセッサから PCAP を使用すると，FPGA の再構成を実行することができる。PCAP インタフェースを使用するには，ソフトウェアから DevC（Device Configuration）ドライバー関数を呼び出す。DevC ドライバは適切な PCAP モードの選択と DMA（Direct Memory Access）転送の開始を管理する。

Zynq-7000 用にビルドされた Linux カーネルには，/dev/xdevcfg というスペシャルファイルが用意されている [14]。xdevcfg は PCAP を制御するためのデバイスドライバで，xdevcfg デバイスに構成データを渡すことで，PL 部の再構成を実行できる。ただし現状ではビットファイルをそのまま渡すことができず，読み込める形式に加工する必要がある。具体的には，ビットファイルのヘッダ部分を除去し，バイトオーダを変更する必要がある。ファイルの加工は Vivado Design Suite 上から Tcl コマンドで行うことができる。

3. Zynq-7000 Soc による実装

〈3･1〉 試作システムの概要 以下，本節では，本研究で試作したシステムの概要について説明する。システムのブロック図を Fig. 3 に示す。本研究の試作システムでは，2 章の巡回動作を再現できるように 4 つの DPR 領域を FPGA 上に配置した。Fig. 3 では，例として 2 種類の回路（WT_1，WT_2）をタイルに配置し，残り 2 枚は予備（Spare）としている。

先行研究 [3]では，耐故障動作を司る RM を FPGA 上に配置し，定期的に再構成することで信頼性を保証しようとした（Fig. 2）。しかし RM の機能が増えると実装が複雑になり，既存の FPGA デバイスで実装すると大きなオーバヘッドを伴う。そこで本研究では，RM 機能を PS 部のソフトウェアとして実現した。PL 部の構成データは SD カード等の外部記憶装置に格納され，PS 部のソフトウェアで主記憶あるいはオンチップ RAM に読み込んでから，PCAP に送られる。

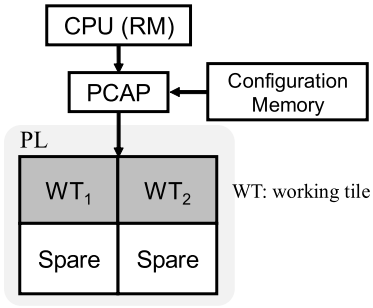

Fig. 3. Block diagram of the proposed system.

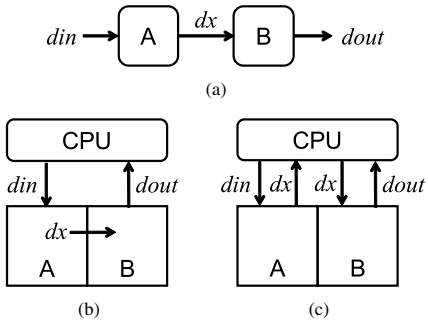

Fig. 4. Logic comprised of two tiles.

本研究では PS 部のハードコアプロセッサで RM 機能を実現したが，ソフトコアプロセッサを利用して RM を PL 上に置き，耐故障動作の対象とすることも考えられる。しかしソフトコアプロセッサは多くの回路リソースを使用し速度も低い。例えば Zynq-7000 にソフトコアプロセッサ（32-bit MicroBlaze）を実装する場合，ルックアップテーブル（LUT）を最小 626～最大 5953 個（ZC7Z010 の 4～34%）消費し，最大動作周波数は 147～265 MHz である [5][15]。一方，本研究で利用した評価ボードでは，PS 部の ARM Cortex A9 は 650 MHz で動作し，浮動小数点演算サポートなど速度面のメリットが大きい。そこで実用的観点から，今回の実装方法を選択した。

ひとつの論理回路を複数タイルで実装する場合，タイル間の論理分割と共に，タイル間のデータ送受が問題となる。Fig. 4 は，複数タイルを使用する論理の例である。入力 din から出力 dout を生成する際，機能を A と B に分割して実装し，A から B へデータ dx を渡すとする（Fig. 4(a)）。これを 2 つのタイルで実装する場合，タイル A からタイル B への配線 dx が必要になる（Fig. 4(b)）。A と B が固定されていれば配線込みの構成データを作成すればよいが，タイルを巡回再配置すると AB 間の位置関係が変わる可能性がある。

この問題に対しては，幾つかの解決法が考えられる。

Table 1. Evaluation environment.

FPGA board	Digilent Zybo
FPGA device	Xilinx XC7Z010-CLG484
OS	Petalinux
CAD	Xilinx Vivado Design Suite (2017.4) Xilinx SDK (2017.4)

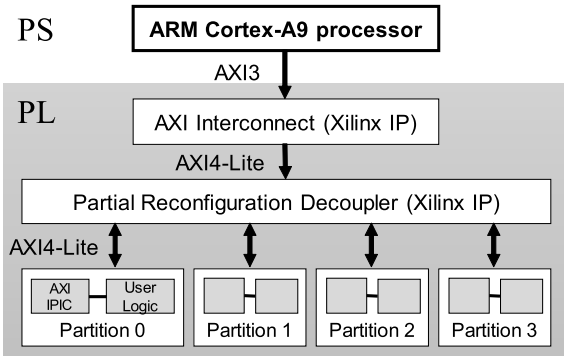

Fig. 5. Hardware design.

- 再配置の際，A と B の位置関係は変えない（マクロ化）
- A と B の可能な全ての位置関係について構成データを用意しておく
- タイル間に Network-on-Chip（NoC）[16]を実装する
- タイル間の通信は全て CPU を経由する（集中化）

本研究の実装では，柔軟性と単純性を重視して，集中化を採用した（Fig. 4(c)）。集中化により，A と B は任意のタイルに配置することができ，また計算の途中結果 dx が CPU に残るので再配置や障害時の復旧が容易になる。より洗練された解決法については今後の課題とする。

〈3·2〉 ハードウェア構成 本研究の実装環境を Table 1 にまとめる。評価ボードには，XC7Z010 を搭載した Digilent Zybo を使用した。PS 部には ARM Cortex A9 が 2 つ搭載されており，OS として Linux（Petalinux）が動作する。PS 部で動作するソフトウェアは，ホスト PC 上のクロス開発環境（Ubuntu 14.04）で開発している。PL 部の論理設計はホスト PC（Windows 10）上の Vivado で行い，電源投入時に実行されるブートローダはホスト PC 上の Xilinx SDK で作成した[17]。

試作システムのハードウェア構成を Fig. 5 に示す。先行研究[3]におけるタイルを再現するため，PL 上に 4 つの DPR 領域（パーティション）を作成した。Fig. 5 におけるパーティション 0〜3 が，Fig. 3 における 4 枚のタイルに相当する。動的再構成が実行されるのは，各パーティションの User Logic の部分である。4 つのタイルは AXI4-Lite で AXI Interconect ブロックに接続され，PS のメモリ上にマッピングされている。各タイルは独立しており，非同期的に動作する。Partial Reconfiguration Decoupler は，未使用のパーティションを切り離して再構成中の誤動作を防ぐために必要である。

PL 部に追加したハードウェアを Linux に認識させるには，作成したハードウェアをデバイスツリーに登録しなければならない。デバイスツリーファイルに各パーティションのノード名と入出力ポートを記述し，Linux ブート時に読み込ませる。Linux カーネルやデバイスツリーなどは，SD カードに格納して評価ボードに挿入しておくと，電源投入時に BootROM によって自動的に読み出される[13]。本研究で作成したファイルを Table 2 にまとめた。

Table 2. List of boot files.

Filename	Description
BOOT.bin	Boot image, including FSBL, u-boot, and bitstream
devicetree.dtb	Device tree
uImage	Linux kernel for ARM
uramdisk.image.gz	Disk image for Linux

4. 応 用 例

本章では，RM をソフトウェア化したシステムの応用例を 2 つ紹介する。

〈4·1〉 複数タイルの使用と巡回動作 まず，複数タイルを使用してタイル間のデータ転送が正しく行われるか確認した。そのブロック図は Fig. 4(c) に示した通りである。PS 部および各タイルは独立に動作するので，データ転送には同期が必要になる。本研究ではバス上でハンドシェークを行い，正しくデータ転送が行われることを確認した。タイル上には加算器や乗算器を実装し，演算結果を予想値と比較して正常動作を確認した。

次にタイル上での巡回再配置を行った。Linux から devcfg 経由で部分再構成を行い，再配置後の回路が正常に動作することを確認した。

〈4·2〉 多重化回路による耐故障化 〈4·1〉節で基本動作を確認したので，次に代表的な耐故障化手法である多重化が提案手法で実現可能であることを確認する。多重化回路は，等価な回路を複数実装して機能に耐故障性を付加する手法である[18]。二重化して出力を比較すれば誤動作を検出することができ，三重化して出力の多数決をとれば単一故障まで検出・修正することができる。

Fig. 6 に，本実装のブロック図を示す。本研究の評価システムにはタイルが 4 つあるので，3 つのタイルに同一の機能（Func）を実装する。残り 1 つはスペアとして未使用（blank）にしておく。CPU は 3 つの Func に同じ入力データを書き込み (a)，それぞれの出力 (b)，(c)，(d) を読みだして結果を比較する。不一致があれば多数決を取り，誤動作したタイルの復旧を試みる。

さらにファームエラーを避けるため，一定時間ごとに構成データをリフレッシュした。更新時，未使用のタイルには Func の構成を書き込み，一番古い Func にブランキングビットストリーム[11][12]を書き込んで消去する。再構成の順番を巡回的に決めることにより，どのタイルも定期的にリフレッシュされる。

以上の動作を実装した結果，三重化回路が正しく動作し，

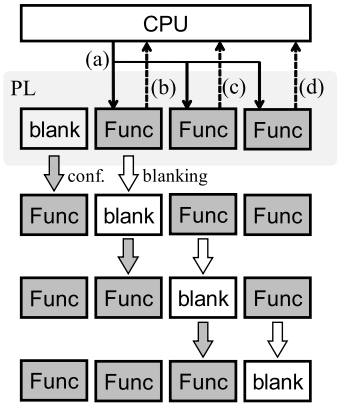

Fig. 6. Example: Triple modular redundancy.

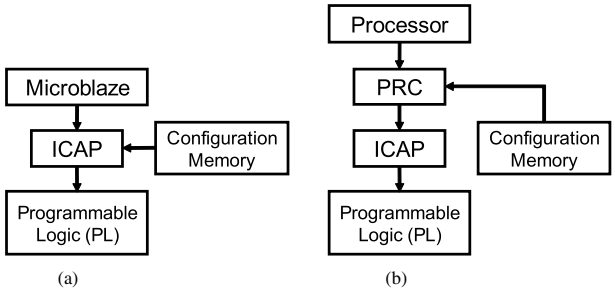

Fig. 7. ICAP and PRC.

同じ結果を返すことが確認された。また巡回的再配置も正常に機能した。短時間の試験では不一致は検出されなかったが，提案手法で三重化回路が実現できることが示された。

本研究の実装では，短い間隔で再構成することにより，単一故障・多重故障に遭遇することなく三重化回路が動作した。再構成の間隔は故障率に合わせて決める必要があるが，この点については使用環境や応用の性質にも依存するため，今後の課題とする。

5. 検　討

〈5·1〉 他のデバイスへの適用可能性　本研究では，Zynq-7000 SoC デバイスに特化した実装を行ったが，提案手法自体は他のデバイスにも適用できるものであり，一定の汎用性を持っている。Xilinx 社のデバイスでは，Zynq-7000 以外にも，多くの 7 シリーズ（Virtex-7，Kintex-7，Artix-7）や Ultrascale デバイスで DPR がサポートされている [11]。Altera（Intel）デバイスにおいても，部分再構成機能は提供されている [4]。

本研究では Zynq-7000 の ARM Cortex A9（ハードコアプロセッサ）と PCAP を用いて実装を行ったが，他のプロセッサあるいはソフトコアプロセッサでも DPR は実現できる。例えば Microblaze ソフトコアプロセッサ [15]であれば，Internal Configuration Access Port（ICAP）を用いて PL 部を DPR することができる [11]。Microblaze は ICAP と直結可能であるが（Fig. 7(a)），他のプロセッサから ICAP を使用する場合は Partial Reconfiguration Controller（PRC）[12][19] を併用する必要がある（Fig. 7(b)）。紙数の関係で省略するが，Zynq-7000 上のハードコア+PRC+ICAP という組合せの実装も行い，DPR の動作を確認済である。

〈5·2〉 再構成時間とデータ転送時間　本手法を実応用に適用する際には，性能面の検討が必須となる。信頼性向上のために動的再構成や巡回再配置を行えば，その処理時間が必要になり，応用の性能を低下させる可能性がある。また，PS 部と PL 部の間でデータの送受が必要になるため，データ転送時間も性能に影響を及ぼす可能性がある。

再構成時間とデータ転送時間はシステムの性能に直接影響するため，本実装手法を他の応用に適用可能か判断する上で重要な基礎データとなる。例えば制御ソフトウェアの一部を高位合成によりハードウェア化して Zynq-7000 で実装した場合 [7]，制御論理の性能自体はハードウェア化により 10 倍向上するが，システム全体の性能は元の 2%～25%に低下することが報告されている。性能低下の要因は，プロセッサと周辺回路（ハードウェア化制御論理）の間のデータ転送である。

データ転送時間 δ は，転送速度 α，遅延時間 β，転送量 s から，$\delta = s/\alpha + \beta$ で表現される。しかし本研究の実装では DMA は使用しておらず，転送は常に 1 ワード（32 bit）単位である。そのため速度 α と遅延 β は分離できず，測定値は 1 ワード転送する時間 δ である。また転送方向（read, write）により時間が異なる可能性があるが，ここではタイルへの送信～折り返し～タイルからの受信という一連の合計時間を測定し，その時間をデータ転送回数で除して δ を求めた。〈4·1〉節で述べた通りハンドシェークを行っているので，その手順も含めると，データ転送回数は 5 回（read 2 回，write 3 回）である。

同様に，再構成時間 γ もサイズに依存すると考えられるが，パーシャルビットファイルのサイズはパーティションを作成した時点で決まってしまう。色々な論理回路でパーシャルビットファイルを作成してみたが，論理構成の内容によらずサイズは一定（本実装では 99772 バイト）であった。パーティションサイズの変更は今後の課題とし，本節では本実装における再構成時間を測定した。

再構成はおおよそ以下の手順で行われる。

(a) SD カードからパーシャルビットファイルを read して主記憶上のバッファに格納する。
(b) /dev/xdevcfg を open する。
(c) xdevcfg にバッファの内容を write する。
(d) /dev/xdvcfg を close する。

再構成時間についても，処理のどこまで含めるかによって，定義が変わる。ここでは以下の 3 通りについて測定した。

γ_1　手順 (a)～(d) の時間。
γ_2　手順 (b)～(d) の時間。
γ_3　手順 (c) の時間。

これらの測定結果を Table 3 にまとめた。

測定値 γ_1 と γ_2 の差から，パーシャルビットファイルの

Table 3. Data transfer time and reconfiguration time.

Data transfer time	δ	336 ns
Reconfiguration time	γ_1	8.8 ms
	γ_2	8.4 ms
	γ_3	4.9 ms

読み込み時間は 0.4 ms と短いことがわかる。また γ_2 と γ_3 の差から，デバイスドライバの処理は 3.5 ms であることがわかる。Kohn の報告[13]では，134 kbyte のパーシャルビットファイルの部分再構成時間は 2 ms となっている。ハードウェア設計およびソフトウェアの相違はあるが，γ_3 は Kohn の報告の倍以上である。この点から見て，再構成時間にはまだ改善の余地があるものと考えられる。適用対象（応用）や使用環境（故障率）により再構成時間への要求仕様は異なるが，本節の結果はその判断の基礎データとして有用であると考えられる。

6. おわりに

本研究では，著者らが過去に提案した手法[3]を実現する手段として，耐故障動作を制御する RM（Recovery Manager）をソフトウェア化し，Xilinx Zynq-7000 SoC を用いて実現した。本研究の実装では，SoC 上で動作する Linux から xdevcfg デバイスを経由して動的部分再構成を制御した。提案手法の基本動作は実機で確認され，多重化回路を用いた耐故障化手法を実装して正常動作を確認した。

本研究では，過去に提案した耐故障化手法[3]の全てを実装評価するには至らなかった。今後は，本研究で実装したソフトウェア RM を利用して，Voting circuit を使用した耐故障化手法を実装したい。

また，今後は本実装の成果を利用して，ソフトウェアのハードウェア化による知的財産保護手法[6][7] を実装・評価する予定である。

謝　辞

本研究の一部は JSPS 科研費 20K11733 の支援による。

文　献

（1） H. Kawai, Y. Yamaguchi, and M. Yasunaga: “Realization of the sound space environment for the radiation-tolerant space craft”, Proc. 2006 IEEE Intl. Conf. Reconfigurable Computing and FPGA, pp.198–205 (2006)
（2） A. DeHon and N. Mehta: “Exploiting Partially Defective LUTs: Why You Don’t Need Perfect Fabrication”, Proc. 2013 Intl. Conf. Field-Programmable Technology (2013)
（3） S. Ogido, C. Yamada, K. Miyagi, S. Ichikawa, and N. Fujieda: “Research on Fault Tolerant Processor using Dynamic Reconfiguration”, IEEJ Trans. IA, Vol.139, No.2, pp.187–192 (2019) (in Japanese)
荻堂盛也・山田親稔・宮城　桂・市川周一・藤枝直輝：「部分再構成を用いたプロセッサの耐故障化手法に関する検討」, 電学論 D, Vol.139, No.2, pp.187–192 (2019)
（4） K. Vipin and S.A. Fahmy: “FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications”, ACM Computing Surveys, Vol.51, No.4, Article 72 (2018)
（5） Xilinx Inc: “Zynq-7000 SoC Data Sheet: Overview”, DS190, v1.11.1, p.3 (2018)
（6） S. Yamada, S. Ichikawa, and N. Fujieda: “Implementation of Obfuscated Control Logic Circuit with LegUp and oLLVM”, IEEJ Trans. EIS, Vol.139, No.9, pp.952–957 (2019) (in Japanese)
山田翔太朗・市川周一・藤枝直輝：「LegUp と oLLVM による難読化制御論理回路の実装」, 電学論 C, Vol.139, No.9, pp.952–957 (2019)
（7） N. Fujieda, S. Ichikawa, Y. Ishigaki, and T. Tanaka: “Evaluation of the hard-wired sequence control system generated by high-level synthesis”, Proc. 26th IEEE Intl. Symp. Industrial Electronics (ISIE 2017), pp.1261–1267 (2017)
（8） S. Ogido, S. Ichikawa, N. Fujieda, C. Yamada, and K. Miyagi: “Implementing Fault Tolerance Method Using Dynamic Partial Reconfiguration by Xilinx Zynq-7000 SoC”, The Papers of Technical Meeting on Innovative Industrial System, IEE Japan, IIS-20-019 (2020) (in Japanese)
荻堂盛也・市川周一・藤枝直輝・山田親稔・宮城　桂：「動的部分再構成を用いた耐故障化手法の Xilinx Zynq-7000 SoC による実装」, 電学次世代産業システム研究会, IIS-20-019 (2020)
（9） Y. Li, D. Li, and Z. Wang: “A new approach to detect-mitigate-correct radiation-induced faults for SRAM-based FPGAs in aerospace application”, Proc. IEEE 2000 National Aerospace and Electronics Conference, pp.588–594 (2000)
（10） V. Lakamraju and R. Tessier: “Tolerating operational faults in clusterbased FPGAs”, Proc. ACM/SIGDA Eighth Intl. Symp. Field Programmable Gate Arrays, pp.187–194 (2000)
（11） Xilinx Inc: “Vivado Design Suite User Guide—Partial Reconfiguration”, UG909 (v2019.1) (2019)
（12） Xilinx Inc: “Vivado Design Suite Tutorial—Partial Reconfiguration”, UG947, v2015.4, p.24 (2015)
（13） C. Kohn: “Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices”, XAPP1159 (v1.0), Xilinx. Inc. (2013)
（14） C. Kohn: “Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite for Zynq-7000 AP SoC Processor”, XAPP1231 (v1.1), Xilinx Inc. (2015)
（15） Xilinx Inc: “MicroBlaze Processor Reference Guide”, UG984 (v2019.1) (2019)
（16） S. Werner, J. Navaridas, and M. Lujan: “A Survey on Optical Network-on-Chip Architectures”, ACM Comput. Surv., Vo.50, No.6, Article.89 (2017)
（17） Xilinx Inc.: “Zynq-7000 All Programmable SoC Software Developers Guide”, UG821 (v12.0) (2015)
（18） 向殿政男：フォールト・トレラント・コンピューティング, 丸善 (1989)
（19） Xilinx Inc: “Partial Reconfiguration Controller v1.3”, PG193 (April 4, 2018) (2018)

荻 堂 盛 也（非会員） 2018 年沖縄工業高等専門学校創造システム工学専攻修了。同年豊橋技術科学大学大学院電気・電子情報工学専攻博士前期課程入学。2020 年 3 月同大学大学院電気・電子情報工学専攻博士前期課程修了。同年 4 月より沖縄電力（株）。

市 川 周 一（正員） 1985 年東京大学理学部卒業。1987 年東京大学大学院理学系研究科修士課程修了。1987 年新技術事業団，1991 年三菱電機（株），1994 年名古屋大学工学部助手。1997 年豊橋技術科学大学工学部講師。同助教授，准教授を経て，2011 年沼津工業高等専門学校制御情報工学科教授。2012 年より豊橋技術科学大学大学院工学研究科教授。現在に至る。理学博士。IEEE（senior member），電子情報通信学会（シニア会員），ACM，情報処理学会，各会員。

藤 枝 直 輝（非会員） 2013 年東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程修了。博士（工学）。同年より豊橋技術科学大学電気・電子情報工学系助教。2019 年より愛知工業大学工学部電気学科講師。プロセッサアーキテクチャ，FPGA 応用，組み込みシステム，セキュアプロセッサの研究に従事。情報処理学会，電子情報通信学会，IEEE 各会員。

山 田 親 稔（上級会員） 2000 年琉球大学大学院理工学研究科博士前期課程修了。2004 年同大学大学院博士後期課程単位取得満期修了。同年拓殖大学北海道短期大学専任講師。2007 年沖縄工業高等専門学校情報通信システム工学科助教。2009 年同高等専門学校情報通信システム工学科准教授。2014 年ビクトリア大学（カナダ）客員研究員。2015 年沖縄工業高等専門学校情報通信システム工学科准教授。2020 年同高等専門学校情報通信システム工学科教授。現在に至る。博士（工学）。形式的設計検証，リコンフィキャラフルシステムの研究・教育に従事。IEEE，電子情報通信学会，情報処理学会，各会員。

宮 城 桂（非会員） 2008 年高知工科大学情報システム工学科卒業。2010 年同大学大学院修士課程修了。2014 年同大学大学院博士課程修了。同年沖縄工業高等専門学校情報通信システム工学科助教。2020 年同高等専門学校情報通信システム工学科講師。現在に至る。博士（工学）。自己同期型回路を用いた超低消費電力 VLSI の研究に従事。電子情報通信学会会員。