システムＬＳＩ設計技術　99－2
（2001．　1．11）

マルチコンテキストFPGAにおける構成情報読み込み時間短縮の検討

北岡 稔朗† 宇野 正樹† 柴田 裕一郎† 天野 英晴†

†慶應義塾大学 理工学部 情報工学科
〒223-8522 横浜市港北区日吉3-14-1
E-mail: {kitaoka, uno, shibata, hunga}@am.ics.keio.ac.jp

あらまし

FPGA、CPLDなどのプログラム可能なデバイスの技術的発展は目覚しく、計算機システムの分野にも大きなインパクトを与えている。しかし、動的再構成可能なFPGA/CPLDは増えているものの、再構成のための構成情報を読み込むためには、多大な時間が必要になる。本論文では、マルチコンテキストFPGAのコンフィギュレーション時間を短縮するために、構成面の1つに、圧縮された構成情報を伸張する機構を設ける手法を検討する。シミュレーションの結果、圧縮率が高い構成情報の場合復号化回路が外部メモリへのバスクロックの2倍で動けばその効果があらわれることが分かった。

キーワード　マルチコンテキスト、可変構造システム、データ圧縮

Reducing the loading time of configuration data on multi-context FPGAs

T. KITAOKA† M. UNO† Y. SHIBATA† H. AMANO†

†Dept. of Information and Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama 223-8522, Japan
E-mail: {kitaoka, uno, shibata, hunga}@am.ics.keio.ac.jp

Abstract

FPGAs (Field Programmable Gate Arrays) have made possible new varieties of reconfigurable systems in which an algorithm is executed directly in hardware. However, although dynamically reconfigurable multi-context FPGAs have been implemented, it takes long time to load configuration data for reconfiguration. In order to reduce the time required for loading configuration data on multi-context FPGAs, we propose configuring a decode function on one context of the FPGA that restores the configuration data compressed in advance. Simulation results show that the decode circuits which operates with 2 times higher frequency compared to the bus clock to external configuration memory reduces the configuration time when the configuration data are well compressed.

key words　Multi-Context FPGA, Reconfigurable System, Data Compression

1 はじめに

FPGA (Field Programmable Gate Array)、CPLD (Complex Programmable Logic Device) などのプログラム可能なデバイスの技術的発展は目覚しく、計算機システムの分野にも大きなインパクトを与えている。中でもアルゴリズムを直接 FPGA 上のハードウェアとして実現する **Reconfigurable System: 可変構造システム**あるいは **Custom Computing Machine** は、従来の計算機とは全く異なった原理に基づく計算システムとして、最近ますます広く研究されている [1]。これらのシステムは、対象とするアルゴリズムをそのままハードウェア化して FPGA/CPLD 上で処理を行う。それぞれのアプリケーションで最適なハードウェア構成をとることで、専用の ASIC に比べて高い柔軟性を持つと共に、汎用 CPU や DSP のようなソフトウェアにより処理を行うシステムに比べて高い性能を得ることを狙っている。すなわち、アプリケーション専用ハードウェア並の性能と汎用計算機の持つ柔軟性を兼ね備えたシステムとして期待されている。

しかし Reconfigurable System は、対象となる問題の規模が大きくなり、必要なハードウェア量がシステムのサイズを超えると、全く計算が不可能になってしまうという汎用システムとして致命的な問題点がある。これを解決するためには、システムの動作中に、ハードウェア構成を動的に入れ替えることで、システムの最大サイズを越えるハードウェアを仮想的に実現する機構が必要になる。このような考え方を、仮想記憶 (Virtual Memory) に倣って、仮想ハードウェア (Virtual Hardware) と呼ぶ。動的再構成可能な FPGA/CPLD は増えているものの、再構成のための Configuration Data を内部の Configuration Memory に読み込むためには、多大な時間が必要になる。

本論文では、あらかじめ圧縮した Configuration Data を FPGA 内で復号することにより、再構成のための時間を短縮する方式を提案し、評価する。この技術は、チップ内に複数の Configuration Memory を持つマルチコンテキスト FPGA で用いると、復号のためのハードウェアを無駄にせずに Configuration Data の高速読み込みが可能になる。

2 コード圧縮による Configuration 高速化

2.1 基本方式

現在の一般的に用いられている SRAM 型の FPGA/CPLD は、Configuration Data が、場合によっては、$5Mbit$ 以上にもなり、設定には、msec オーダーの時間を要する。これは、現在の用途では、Configuration Data の設定は、システム立ち上げ時にのみ行うため、設定時間の短縮よりも、設定回路の容易さに重点が置かれた設計になっている。このため、設定の周波数は $10MHz$ 程度になっている。動的再構成用に、設定時間を現在の一般的な RAM と同程度の速度にすることは技術的には充分可能である。50MHz 程度の周波数で 64bit 単位で設定を行う高速設定モードを設けたとしても、$5Mbit$ の Configuration Data を設定するのに $1.6msec$ かかってしまう。

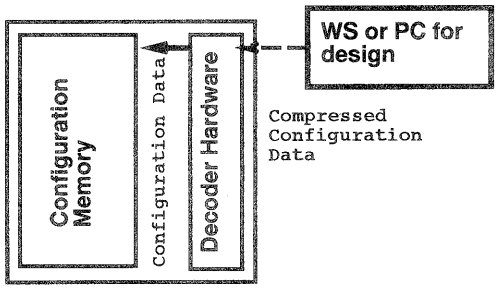

図 1: コード復号回路付き FPGA

そこで、コード圧縮手法を用いて Configuration の高速化を行う手法を提案する。この手法では、Configuration Data は、配置配線後に、あらかじめソフトウェアにより圧縮された形で用意する。FPGA/CPLD 内にはコード復号用ハードウェアのみを持ち、このハードウェアが On-the-fly に流し込んだデータを復号して、FPGA 中の Configuration Memory に書き込む。しかし、この方法では、図 1に示すように FPGA 内に復号用のハードウェアを常に保持する必要があり、このためのコストが大きくなる問題点がある。

2.2 マルチコンテキスト FPGA との組み合わせ

この問題点は、最近現実化されてきたマルチコンテキスト FPGA を用いることにより、解決することができる。

マルチコンテキスト FPGA は、図 2 に示すように、デバイス内部の Configuration Memory を複数セットに拡張し、これにあらかじめ必要な Configuration Data を格納しておく。それぞれの Configuration Memory の出力をマルチプレクサで切り替えることにより、デバイス上の回路を高速に入れ換えることが可能となる。それぞれの Configuration Memory 上の配線情報をコンテキストと呼び、マルチプレクサによるコンテキストの切り替えをコンテキストスイッチと呼ぶ。コンテキストスイッチは多くのチップでは、1 クロックで可能であることからマルチコンテキスト型 FPGA は、高速にハードウェア構成を入れ替えることが可能である。マルチコンテキスト FPGA は、既に NEC により試作され

たDRL[2]が利用可能であり、仮想ハードウェアWASMIIがこの上に実現されている[3]。

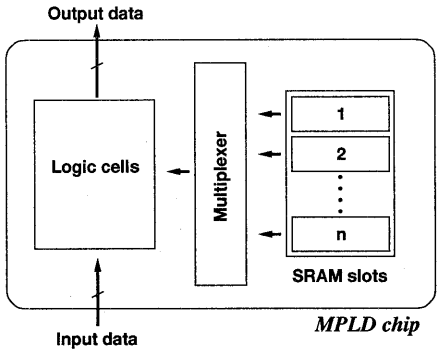

図2: マルチコンテキストFPGA

高速Configurationを行う場合、あらかじめ、このうちのコンテキストの1つに復号用ハードウェアのConfiguration Dataを設定しておく。そして、Configuration時には、このコンテキストを用いることで、他のコンテキストに高速にConfigurationを設定し、終了後速やかにコンテキストを入れ替えて処理を開始する。

さらに、将来、部分再構成可能で多くのコンテキストを持つDRAM型マルチコンテキストFPGA[4]が登場した場合、それぞれのブロック毎のConfigurationが可能になる。この場合、図3に示すように、必要に応じて、一つのブロックに復号用ハードウェアのコンテキストを設定し、他のブロックの使っていないコンテキストのConfigurationを処理中に書き換えることが可能になる。ブロック毎のコンテキストの書き換えをスケジュールすることにより、ほとんど外部からのConfiguration時間を隠蔽することができ、外部に多量のコンテキストを置いておき、必要な際にConfigurationを行う仮想ハードウェアの実現を可能にすることが期待できる。

本手法で鍵となるのは、Configuration Dataの圧縮と復号用のハードウェアである。しかし、今までConfiguration Dataの圧縮／復号は、高速Configuration自体がほとんど考えられていなかったこともあり、ほとんど検討されていない。そこで、本稿では、本方式の実現可能性を検討するため、Configuration Dataの圧縮効果について調査し、復号方式と、そのハードウェアの動作速度およびハードウェア量の評価を行う。

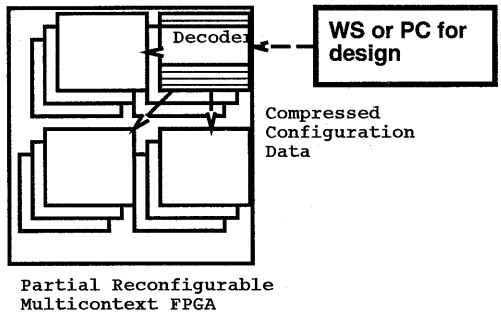

図3: 部分再構成可能なマルチコンテキストFPGA

3 圧縮方式

3.1 符号化法の検討

データ圧縮のための符号化／復号化には様々な方式が用いられている。Configurationの高速化に用いるためには、Configuration Dataを読み込む際にOn-the-flyで、復号することが望ましい。さらに、復号は、FPGA内のハードウェアを用いて行うため、できる限り単純でハードウェア化が容易な方法が望ましい。

そこで、ここでは、Lempel-Ziv符号化法(以下、LZ符号化法[5])のうちLZ77および、Huffman圧縮[10]が考えられる。これらの方法では、圧縮対象のConfiguration Dataを一定のword単位で圧縮するが、今、このword長を8bitとした時の圧縮率を測定した。

測定対象には、NECのマルチコンテキスト型FPGAであるDRL[2]上でVan der Polの常微分方程式を解く回路とNeural Networkを用いてN-Qqueenを解く回路[3]のConfiguration Dataを用いた。この結果を表1に示す。

表1: LZ圧縮とHuffman圧縮の比較

圧縮法	Van der Pol	N-Queen
LZ	68.0%	27.4%
Huffman	51.7%	28.1%

表1のように、圧縮率については、両者ほぼ同程度と考えらる。ただし、Huffman圧縮を復号する回路は、1bitづつ木を辿る必要があるため、回路が複雑化する可能性が高い。そこで、本稿では、LZ77符号を用いることとした。

3.2 符号化の単位と圧縮率

先に触れたように、実際の Configuration Data を LZ 符号化する際には、符号化の単位の word 長を定める必要があり、これが圧縮率に影響を与える。そこで、前節で評価した回路に関して符号化の単位の bit 幅を変化させた場合の圧縮率を調べた。

表 2: 符号化の単位を変化させた場合の圧縮率

bit 幅	Van der Pol	N-Queen
8	68.0%	27.4%
16	66.0%	20.8%
24	57.8%	20.2%
32	66.1%	15.6%
48	56.5%	13.0%
64	57.6%	17.4%
80	52.0%	11.0%
128	49.1%	11.6%

データ圧縮率という観点から見れば、1word を 8bit 幅とするのが最も圧縮率が優れているが、回路構成上は、処理 bit 幅が大きい方が効率的である。そこで、ここでは word 長 8bit 及び 64bit の場合を扱うこととする。

3.3 ゲート利用率の影響

ゲート利用率が Configuration Data の圧縮率にどのような影響を与えるかを調べる。ここでは、現在実装中の並列計算機 SNAIL-2[6] の制御回路、及び RHiNET[8] 用のネットワークインタフェースの制御回路を評価の対象として選んだ。この回路は、Altera 社 Flex10k(EPF10K100)[7] 上に実装された Processor Controller 及び Memory Controller と、Flex10k(EPF10K250) 上に実装された Network Controller の Configuration Data である。

表 3: ゲート使用率と圧縮率との関係

	ゲート使用率	圧縮率
Processor Controller	31%	77.6%
Memory Controller	19%	86.7%
Network Controller	16%	86.4%

表 3からゲート使用率は低ければ低い程、圧縮率が高くなる傾向があることがわかる。

4 LZ 復号化回路

4.1 LZ77 法

LZ77 符号化法は、J.Ziv と A.Lempel によって 1977 年に提案されたデータ符号化法である。この方法は、符号化に用いる辞書を前もって編集せず、符号化する記号系列を読み込みながら辞書を編集しながら符号化する点に特徴がある。

LZ77 符号化法ではスライド窓と呼ばれるバッファを用い、バッファには既に符号化された記号系列を格納する。このバッファは符号化が進むにつれて記号系列上をスライドしていくことから一般にスライド窓と呼ばれる (図 4)。符号化の際にはその時点で符号化されるべき記号系列が、以前に現れてスライド窓に格納されている記号系列であった場合に、(位置, 長さ) のペアで置き換える。例えば "ABCPQABCRS" では 2 回目に現れた ABC は 5 記号前の 3 文字と一致するので ABCPQ(5,3)RS と符号化する。ここでは Ziv-Lempel の論文と若干異なり、次に来るビット列が (位置, 長さ) のペアか符号化されない記号かを区別するために、余分に 1 ビットを付け加える。

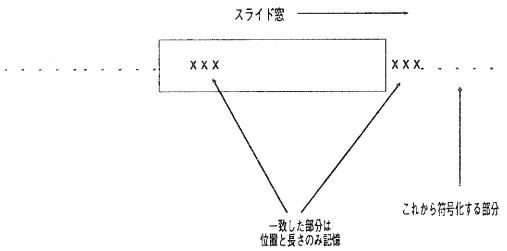

図 4: LZ77 符号化

復号化過程ではバッファに既に復号化した記号系列を格納し、(位置, 長さ) のペアが来たらバッファのその位置からその分の長さの記号系列を出力する。

4.2 復号化回路の設計

この LZ77 の復号化ハードウェアを、Altera 社の FLEX10k 上への実装を想定して設計した。LZ77 復号化を高速ハードウェアで実装する方法として PAHL[9] などが提案されているが、かなり大量のハードウェアが必要である。そこで、ここではマルチコンテキスト型 FPGA に実装することをふまえて、コンテキストに収まるように出来るだけ単純な構成を目指す。ここで実装した復号化回路の構成は図 5のようになっている。

設計の方針は以下の通りである。

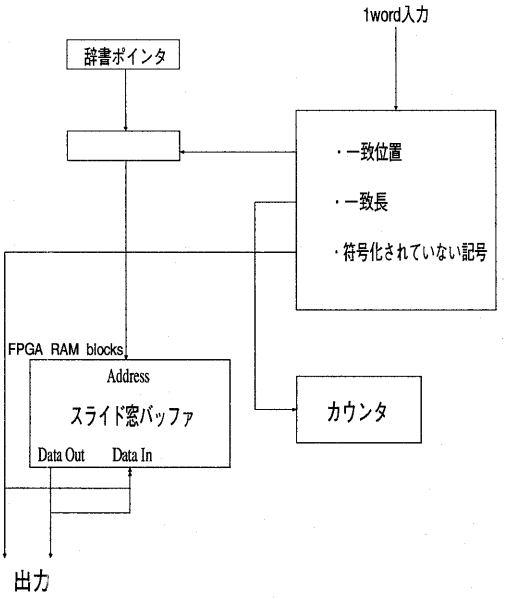

図 5: LZ77 復号化回路

- 入出力は圧縮の単位である word 長幅とし、さらに各記号が符号化されているか否かを示す 1bit の入力を設ける。
- 既に復号化した記号系列を格納するスライド窓バッファを 256word 分持つ
- 符号の伸長の際の出力記号数の制御の為のカウンタ、及びスライド窓バッファの先頭位置を記憶しておくためのレジスタを持つ

復号の際に行われる具体的な処理は以下の通りである。

1. 1word 分データを入力する。符号化されているかどうかの符号 bit によって処理が分かれる。
2. 符号化されていない記号が入力された場合はそのまま出力すると同時に、辞書ポインタの指すアドレスのバッファにその記号を格納し、再び入力を待つ。
3. (位置, 長さ) のペアが来た場合は符号の伸長を開始し、その間は復号化回路への入力を止める。まず長さ分だけカウンタをセットする。カウンタを 1 づつ減らしながら、辞書ポインタから一致位置を減じた値の指すアドレスのバッファから記号系列を 1word づつ出力し、同時にバッファへも追加していく。カウンタが 0 になったら復号終了し再び入力を待つ。
4. 辞書ポインタの値はバッファに追加する度に増やしていく。バッファが一杯になったら最初にもどりバッファの内容は古い方から書き潰されていく。

復号化回路を実現するにあたって最もゲート数を消費するのはスライド窓バッファである。256x8bit 程度のバッファでもレジスタメモリとして実装するのは困難である。しかし、外部メモリとして実装するとするとバンド幅を広くとることが難しくなり動作速度面で不都合であり、又、実際にマルチコンテキスト FPGA に実装する際に問題である。そこでバッファは FLEX10K 内蔵のデュアルポートメモリを用いる。それでも、幾つものデータを同時に読み書きすることが困難であり、同時に複数の符号を処理することが難しくなる。word 長 8bit と 64bit の場合の動作速度とハードウェア量の評価を表 4に示す。

表 4: 動作速度とハードウェア量

bit 幅	8bit	64bit
ゲート数	8233	19812
動作周波数 (MHz)	32.6	14.2

5 結果

前章で実装した復号化回路では、入力 bit 幅と出力 bit 幅が等く、(位置, 長さ) のペアが来た際に入力を止めて 1cycle に 1word づつ伸長していくため、復号化回路と圧縮した Configuration Data を取り込む速度が等しい場合、高速化を実現することは出来ない。それどころか、伸長開始時と終了時に余計な cycle がかかるためかえって遅くなってしまう。しかし、将来的には復号化回路が外部メモリへのバスクロックより速くなるという仮定をすると、このままでも十分高速化が期待できる。

word 長が 8bit と 64bit の各々の場合で、Van der Pol の常微分方程式を解く回路と N-Queen 問題を解く回路の Configuration Data を非圧縮で取り込んだ場合と圧縮して取り込みチップ内で復号化した場合の比較を図 6、7、8、9 に示す。グラフの x 軸は外部から Configuration Data を取り込むクロックと内部の伸長回路のクロックの比率であり、y 軸は取り込むのにかかるクロックサイクル数である。

word 長が 8bit 幅の場合、(位置, 長さ) のペアが来た時に各々を読み込むのに 1cycle づつ要する。しかし word 長が 16bit 以上ある場合、(位置, 長さ) のペアを同時に読み込むことが可能なので、伸張効率が良くなる。Van der Pol の常微分方程式のように、Configuration Data の高圧縮化を実現できた場合、復号化回路が外部メモリへのバスクロックよりも 2 倍高速に動作すれば、高速化が実現できると思

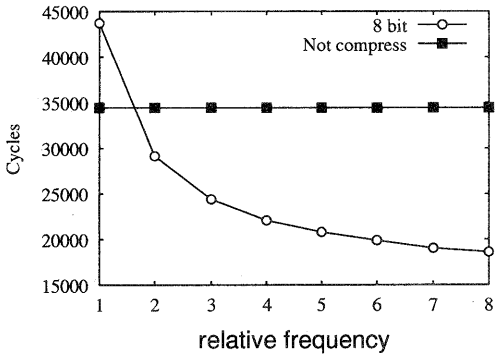

図 6: Van der Pol(word 長 8bit)

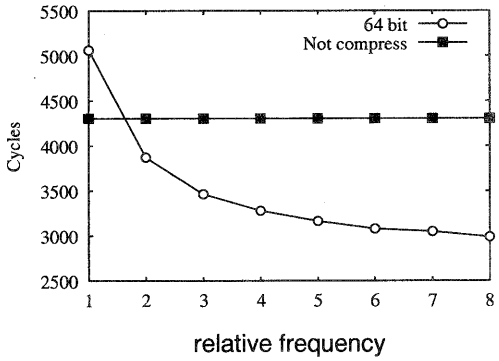

図 7: Van der Pol(word 長 64bit)

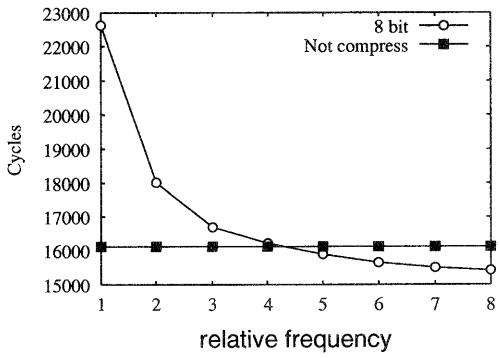

図 8: N-Queen(word 長 8bit)

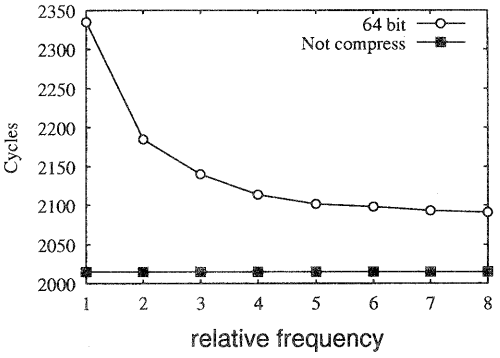

図 9: N-Queen(word 長 64bit)

われる。逆に N-Queen 問題 (word 長 64bit) のように、圧縮後のサイズがそれほど減らなかった場合は復号化回路が高速動作しても、高速化を実現することは困難である。

6 まとめと今後の課題

現状の復号化回路では、Configuration Data の圧縮率が高い場合でないと、高速化の効果が現れにくい。そこで、以下のような改善の方針が考えられる。

- 内部に専用回路を組み込む場合は、今回の評価回路よりも相当高速に動作させることができる。専用回路をデータの転送レートよりも高速に動作させれば、高速復号が容易となる。
- 今回の復号化回路は Altera 社の FLEX10K 上での実装を想定したため、比較的単純な Dual Port RAM を利用している。複数 word を同時に読み書きする数を増やすために 3-port, 4-port のメモリが利用可能であれば、さらに高速化することができる。

以上の方法を用いてさらなる高速化の方法を探る予定である。

参考文献

[1] 末吉敏則, 飯田全広 “リコンフィギュラブル・コンピューティング” 情報処理, vol.40, no.8, 1999

[2] T. Fujii et. al. “A Dynamically Reconfigurable Logic Engine with a Multi-Context/Multi-Mode Unified-Cell Architecture” Proc. International Solid State Circuit Conference 1999

[3] M. Uno, et. al. "Implementation of Virtual Hardware on Dynamically Reconfigurable Logic" Proc. SCI/ISAS 2000.

[4] M.Motomura et. al. "An Embedded DRAM-FPGA Chip with Instantaneous Logic Reconfiguration" Proc. Symposimu on VLSI Circuits 1997

[5] Jcob Ziv and Abraham Lempel. "A universal algorism for sequential data compression" IEEE Transactions on Information Theory, IT-23(3):337-343, 1977

[6] 星野智則, 緑川隆, 金森勇壮, 天野英晴 "キャッシュ制御機構を持つスイッチ結合型マルチプロセッサ SNAIL-2 の評価" 信学技報, vol.100, no.248, CPSY-40-51:pp9-16, 2000

[7] "Altera Data Book" 2000

[8] Tomohiro Kudoh, Shinji Nishimura, Junji Yamamoto, Hiroaki Nishi, Osamu Tatebe and Hideharu Amano, "RHiNET: A network for high performance parallel processing using locally distributed computers", IWIA 99

[9] 藤岡豊太, 阿曽弘具, "高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL の構成" 信学技報, vol.96, no.342, CPSY96-69:1-7, 1996

[10] D.A.Huffman. Proceedings of the Institute of Radio Engineers, 40:1098-1101, 1952