第 4 回 情報輪講 資料

Field Programmable Gate Array を用いた マイクロプロセッサ

田中英彦研究室 博士過程 1 年 吉瀬 謙二

1997 年 11 月 10 日

概要

容易に論理の書き換えが可能なデバイス FPGA(Field Programmable Gate Array) が注目を浴びている。従来は、値段が高い、利用できるゲート数が少ない、動作周波数が低いといった問題点から、利用は非常に制限されてきた FPGA だが、近年のデバイス技術の向上により、先の問題点の多くは改善される方向にある。本輪講では、FPGA の構造と技術動向を解説した後に、具体的な研究例として、動的に再構成可能なマイクロプロセッサを取り上げる。この様な計算機環境では、命令セットアーキテクチャをインタフェースとしたソフトウェアとハードウェアの分離という概念すら通用しなくなる。更に、この変化は、計算機アーキテクチャだけでなくコンパイラや、プログラミングに至るまで多大な影響を与える可能性がある。

1 Field Programmable Gate Array(FPGA)

FPGA は、プログラム可能なゲートアレイである。本章ではゲートアレイの説明から始め、FPGA の構造と技術動向を説明する。

1.1 ゲートアレイの構造

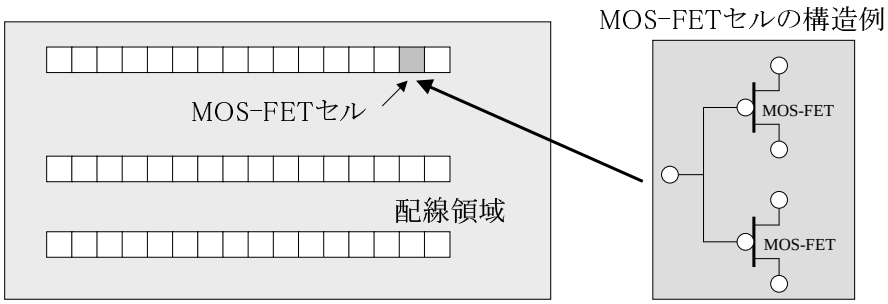


図 1: ゲートアレイの構造

図 1に示す様に、ゲートアレイは MOS-FET(Metal Oxide Semiconductor-Field Effect Transistor) が、NAND や NOR などのゲートを作りやすい配置に数千から数万個並んでいる構造を持つ。論理ゲートが規則的な間隔を持ったアレー状に配置されていることから、ゲートアレイと呼ばれる。MOS-FET が数個集まって、AND や OR, フリップフロップなどのゲートを実現し、さらに、ゲートがたくさん集まって演算

器やレジスタ、メモリ、そして CPU などができあがる。ここで、MOS-FET の構造をユーザ (設計者) が変更することはできない。ユーザが指定することができる部分はトランジスタ (MOS-FET) 間の配線であり、この配線を決定することがゲートアレイの設定となる。

具体的には、ユーザが回路図のかたちで MOS-FET 間の配線を決定するマスクパターンを設計し、工場でマスクパターンに従って IC の製造をおこない、IC の完成品がユーザに届くことになる。

FPGA と比較する上で重要なことは、ゲートアレイ[1] ではトランジスタ・レベル (ゲート・レベル) で接続を指定し、さらにその接続は配線領域を自由に用いることができるということである。

1.2 Programmable Logic Device(PLD)

ゲートアレイと異なり、配線の設定を変更することでプログラム可能なディジタル IC として、PLD または PLA(Programmable Logic Array) と呼ばれる素子が 1970 年代から利用されてきた。

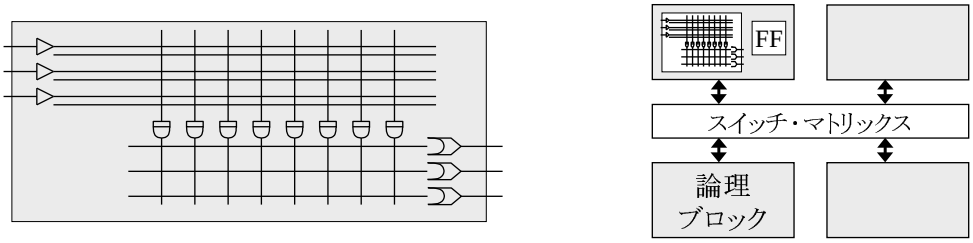


図 2: PLD と CPLD の構造

PLD は、図 2左 に示すように NOT、AND、OR の各ゲートをこの順番に配置し、この間の配線を自由に変えることができる配線マトリックスを持つ。PLD を利用する時には、この配線マトリックスの ON/OFF を指定することで、様々な組合せ回路を表現できる (**プログラム可能**) ことになる。PLD にフリップフロップ (記憶素子) を接続することで、任意の順序回路を構成することができるが、PLD を用いて CPU の様な大規模な順序回路を構成することは困難である。

Complex PLD (CPLD) と呼ばれる IC は、単純な PLD にフリップフロップを加えた回路を**論理ブロック**としてチップ内に複数配置し、それらの論理ブロックを後に述べるスイッチ・マトリックスで接続した構成を取る (図 2 右)。PLD と比較し CPLD では複雑な回路の実現が可能となるが、近年のさらなるデバイス技術の向上により、集積できる論理ブロックの数は増加し、それらの多くの論理ブロックをスイッチ・マトリックスのみで接続することは困難となっている。

1.3 FPGA(Field Programmable Gate Array)

FPGA は 1985 年にザイリンクス社から発売された IC で、図 3 に示す様に、**論理ブロック**、クロスポイントスイッチ、スイッチマトリックスという 3 つの要素ブロックを敷き詰めた構造を持つ。論理ブロックは論理と記憶素子を持ち、クロスポイントスイッチとスイッチマトリックスは論理ブロックの接続に自由度を与える配線領域と考えられる。図 3では、論理ブロック A と論理ブロック E を接続する際に、2 つのクロスポイントスイッチと 1 つのスイッチマトリックスを経由していることを直観的に示している。

CPLD の構造と比較すると、FPGA では配線領域が占める領域が広くなっていることがわかる。これは、配線の自由度を確保するためである。次に、各要素の構造を説明する。

[1] ゲートアレイの他に、標準的なセルを自由に配置できるスタンダードセルや、配置・配線全ての構成を設計できるフルカスタムがあり、商用のマイクロプロセッサの多くは効率のよいフルカスタムで作られている。

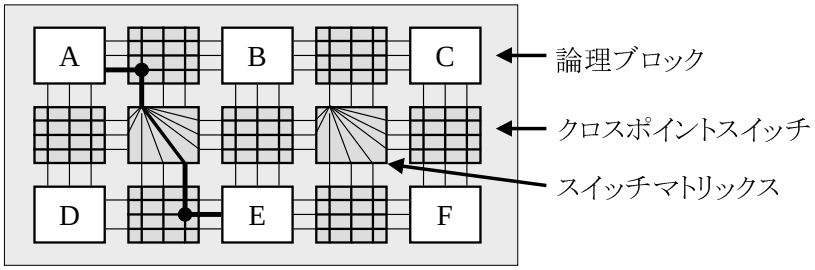


図 3: FPGA の構造例

1.3.1 論理ブロックの構造

図 4 に、論理ブロックの実現例を示した。論理ブロックは、ルックアップテーブルとセレクタ、フリップフロップからできている。

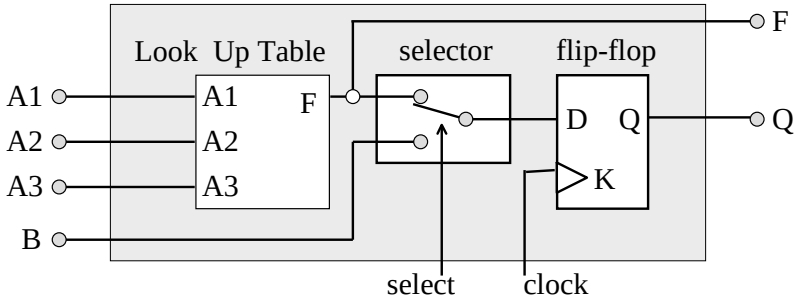


図 4: 論理ブロック

input	A1	0	0	0	0	1	1	1	1
input	A2	0	0	1	1	0	0	1	1
input	A3	0	1	0	1	0	1	0	1
output	F	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

表 1: ルックアップテーブルの真理値表

ルックアップテーブルは表 1の真理値表で示すように、入力の組合せに対して出力の 0/1 を定義することで必要な組合せ回路を実現する。表 1の例では、入力を A1, A2, A3 の 3ビットとしているので、$2^3 = 8$ 通りの入力について出力ビットを定義することで、AND や OR といった論理を実現する。表 2 にルックアップテーブルの設定例を示す。この例では 3 入力の OR を実現する論理ブロックを構成できる。

このルックアップテーブルの内容やセレクタの接続のための情報は、論理ブロック内に配置されている SRAM [2] の各 bit によって保持される。外部からこの内容を設定することで論理ブロックの機能を プログラム することができる。

[2]ルックアップテーブルや接続の情報を記憶する方法として、(1) ヒューズ (2)EPROM(書き換え可能な ROM) (3)SRAM 等が利用されている。ヒューズを用いる方法は、一度切ったら元に戻らないため、再書き込みができないがスイッチでの遅延が最も小さい。EPROM では電源を切っても内容が消えることはないが、内容の書き換えには専用の書き込み器を用いる必要がある。SRAM は電源を切ると内容が消えてしまうが、高速かつ低電圧で内容を変更できるという利点がある。本稿では容易に内容を変更できる SRAM に絞って話を進める。

input	A1	0	0	0	0	1	1	1	1
input	A2	0	0	1	1	0	0	1	1
input	A3	0	1	0	1	0	1	0	1
output	F	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

表 2: 3 入力の OR を実現するルックアップテーブルの真理値表

1.3.2 クロスポイントスイッチ、スイッチマトリックスの構造

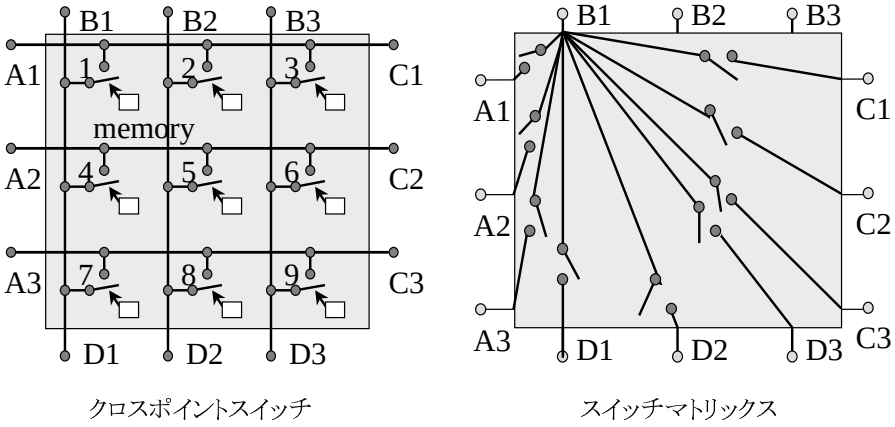


図 5: クロスポイントスイッチとスイッチマトリックスの構成

図 5にクロスポイントスイッチとスイッチマトリックスの構成を示す。クロスポイントスイッチは、配線の交差する点にスイッチを設けたもので縦に伸びる配線と横に伸びる配線を接続する。スイッチマトリックスは、任意の端子と自身以外の全ての端子とをスイッチを介して結んだものである。図 5では、端子 B1 からの接続のみを示し、他の接続は省略してある。FPGA では、これらのスイッチを変更することで論理ブロック間の接続をおこなう。

1.4 IC の分類

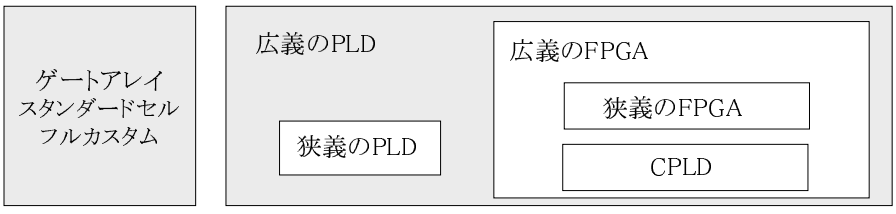


図 6: 混乱をもたらす書き換え可能デバイスの用語

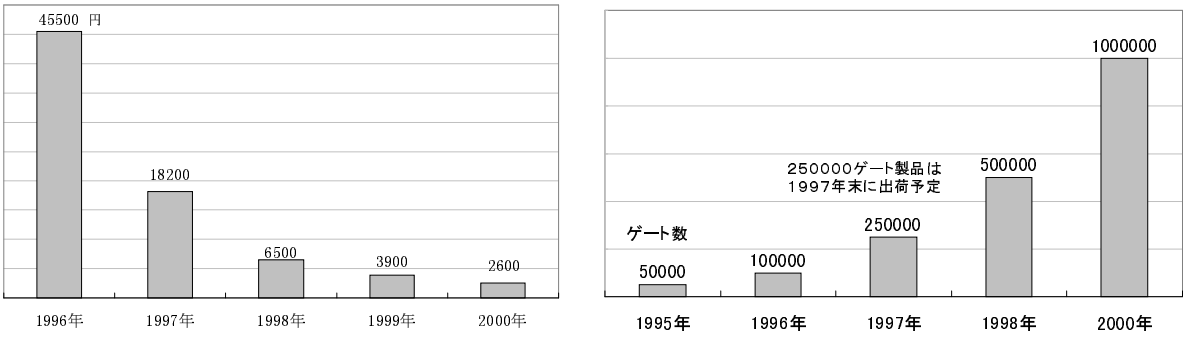
本章で示した IC の関係を図 6に示す。狭義の PLD とは NOT - AND - OR の各ゲートを接続した構成 (図 2) をもつ単純な IC を意味する。狭義の FPGA とは、図 3で示した構成をとる IC を意味するが CPLD と狭義の FPGA を含めて単に FPGA と呼ぶことがある。これは、別々に発展して来た CPLD と FPGA が互いの利点を採り入れ始めたからである。さらに書き換え可能な IC の総称として PLD という名称を用

いることがある。この様に PLD と FPGA は複数の意味を持つため、文脈により意味する所を判断する必要がある。

1.5 PLD の技術動向

従来、PLD の問題点として、利用できるゲート数が少ない、チップの価格が高い、動作周波数が低いといった指摘があった。ここでは、これらの指摘が過去のものとなりつつあることを示す。

図 7に、10 万ゲートの CPLDデバイスの価格推移予測と、PLD の集積度予測を示す。1997 年には 18200 円だったデバイスが西暦 2000 年には 2600 円になると予測されている。PLD の集積度をみると 1997 年に 13 万ゲート製品の出荷が予定されており、2000 年には 100 万ゲートの製品が出荷されると予想されている。



EPK10K(10万ゲート製品)の予定価格　　PLD集積度の動向

図 7: FPGA の価格と集積度の推移 (Design Wave Magazine No.12,1997 より)

この様に PLD の世界では急激な低価格化と集積度の向上が進んでおり、今後もこれらのデバイスが普及すれば、順調に大規模化・低価格化が進むと見られている。このような大規模化は、従来、複数の FPGA を用いて構成していたシステムを 1 チップの FPGA で実現不可とするだけでなく、CPU 等の大規模な回路の実現をも可能にする。

順調に進歩している PLD だが問題点も多い。動作周波数の問題は、配線領域の遅延とルックアップテーブルを引くという PLD の構成上、解決することは困難である。フルカスタムで設計したチップと比較し、5 分の 1 以下の速度でしか動作しないといわれている。

SRAM 等の不揮発性の記憶素子を用いた PLD の場合、記憶素子に設定情報をロードする必要があるが、この情報は膨大なものになる。例えば 2000 ゲートクラスの FPGA デバイス (XC5202) でさえ、構成情報のサイズは約 42Kbit となる。特に動的に構造を変える PLD の場合には、このような情報をどのように供給するかという問題がある。FPGA における接続の自由度の大きさが設定情報を大きくしているため、文献 [5] では FPGA の自由度を制限する方向として Field Programmable Accumulator Array(FPAA) を提案している。これは FPGA における論理ブロックの内容をルックアップテーブルとフリップフロップではなく ALU とレジスタという大きい単位に置き換えたもので、効果的な構成情報の削減を目指している。

本章では、ゲートアレイ、PLD、CPLD、FPGA の構造および、PLD の技術動向を述べた。次章では、FPGA を用いた研究例を紹介する。

2 FPGA を用いたマイクロプロセッサ

2.1 Function Booster Ensemble

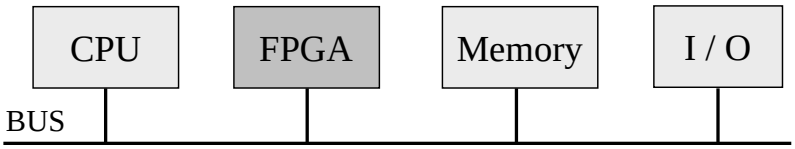


図 8: Function Booster Ensemble

Function Booster Ensemble とは、図 8に示すように、従来の CPU、memory、I/O という構成に FPGA を追加した形をとり、CPU でおこなっていた処理の一部を FPGA で高速処理することで全体としての処理能力の向上を目指す手法である。

ここでは、FPGA を用いたマイクロプロセッサの例として、Sweden Royal Institute of Technology でおこなわれている研究を紹介する [1]。この研究では、C で書かれたプログラムをコンパイラが自動的に FPGA で処理する部分と CPU で処理する部分に分割するという立場をとっている。

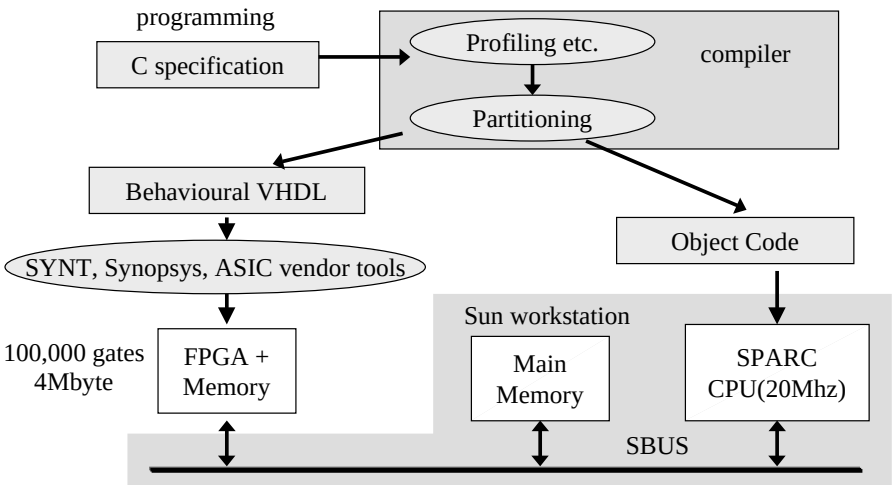


図 9: Design Flow of Function Booster Ensemble

プログラムが実行される様子を図 9 に示す。C 言語で書かれたプログラムは、変更を加えた GNU CC コンパイラにより、ハードウェア・パート (FPGA が実行する部分) とソフトウェア・パート (CPU が実行する部分) に分割される。ハードウェア・パートは自動的にハードウェア記述言語 (Hardware Description Language) に変換された後、FPGA の統合環境 (SYNT, Synopsys, ASIC vender tools) を用いて FPGA の設定ファイルを作成する。ソフトウェア・パートは通常通りにコンパイルされ、実行コードに変換される。コンパイラがプログラムをどのようにハードウェア・パートとソフトウェア・パートに分割するかは 2.1.1節において説明する。

図 10左下の FPGA とローカルメモリは、ワークステーション (SPARCstation IPX 20MHz) のシステム・バスに接続されるボードの上に配置されている。

評価システムにおける FPGA のボード構成を図 10 に示す。XC4005 が、ハードウェア・パートを実行する FPGA でありデータバスとアドレスバスにより 2 つのローカルメモリに接続されている。XC4003 は、

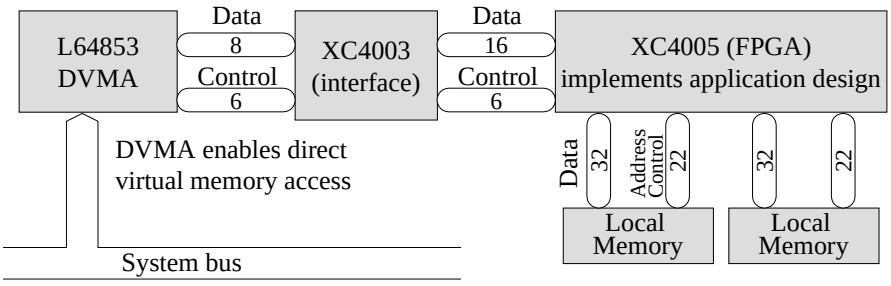


図 10: Board architecture

XC4005 と DVMA とのインタフェースとなるチップである。DVMA はシステム・バス経由でメイン・メモリに直接アクセスをおこなうためのチップである。各チップは、バス上に示したビット数で接続されている。試作で用いている FPGA としては数万ゲートの実現が可能なチップを用いている。(なお、1997 年 11 月 5 日現在、11XC4000 シリーズの最新チップで利用できるゲート数は 5000 から 85000 ゲートとなっている。http://www.xilinx.com)

2.1.1 ハードウエア・パートとソフトウエア・パートの分離

プログラムの FPGA への切り分けは、コンパイル時にプログラムを部分的に実行して得られるプロファイル情報と、性能向上の見積りをもとにおこなっており、以下の 3 つの作業から成る。

1. 候補の切り分け
   基本的にはインナー・ループを FPGA 実行の候補として選択する。ただし、ライブラリ関数を含んだ部分、浮動小数点を含んだ部分は候補としないという制限がある。前者の制限は、分離に必要な情報をライブラリとなったコードから得られないこと、後者は浮動小数点の演算を FPGA に実装することはゲート数の点から困難だったためと考えられる。

2. データのメモリへの割り付けと性能向上の見積り
   FPGA から直接メインメモリにアクセスできるが、このアクセス時間は CPU が用いるキャッシュと比較すると非常に遅い。このため、FPGA の実行においては、次の 3 つに記憶階層を想定して、できる限り高速な記憶階層にデータを割り付ける。

   (a) FPGA 内に存在するデータへのアクセス

   (b) ローカル・メモリ (4MB) へのアクセス

   (c) メイン・メモリへのアクセス

   データをそれぞれの記憶階層へ割り付けた後に、候補の実行時間の見積もりをおこなう。

3. 候補の FPGA への割り付け
   (1) で選択され、(2) で実行時間の見積もりが終った候補の中から、性能向上をもたらす候補を FPGA 内に割り付けていく。

2.1.2 評価環境

プログラムにとって、ハードウェア部分の呼出は関数呼出しと同様におこなわれる。すなわち、呼出し時のレジスタの値が渡され、また変更された値がレジスタに戻される。

	FPGA - main memory	FPGA - local memory
Architecture1 (A1)	500ns	100ns
Architecture2 (A2)	20ns	20ns

表 3: Memory access time

評価には表 3 に示す 2 つの設定を用いている。A1 という設定はプロトタイプ・アーキテクチャにおける設定であり、A2 という設定は FPGA が CPU と同様にメモリにアクセスできると仮定した場合の設定である。A2 における評価ではプロトタイプではなくソフトウェア・シミュレーションにより結果を得ている。評価関数としては性能向上、利用した FPGA のゲート数の他に次に定義する temporal size と max speedup を用いる。

temporal size プログラムの全実行時間に対して FPGA 内で実行された時間の割合を temporal size と定義する。実行が全て CPU でおこなわれた場合に temporal size が 0%となり、実行が全て FPGA でおこなわれた場合に 100%となる。

max speedup FPGA 内の実行時間を 0 と仮定した場合の性能向上率を max speedup と呼び、先に定義した temporal size を用いて次式で表現できる。

$$max\ speedup = \frac{100}{100 - temporal\ size}$$

max speedup は、実行が全て CPU でおこなわれた場合 (temporal size = 0) に 1 となり、実行が全て FPGA でおこなわれた場合 ∞ となる。

2.1.3 評価結果

評価には表 4 に示す 8 つのプログラムを用いている。filter, filter2 は FIR filter、hamming は 100 万ペアに対するハミング距離の計算、sed はストリーム・エディタ、grep と egrep はパターン・マッチング、gzip はファイルの圧縮、gs はポストスクリプトを表示する。ただし、filter2 では、ライブラリ関数と浮動小数点の演算を排除することでハードウェア・パートに分割する際の制約を排除している。その点を除いて filter1 と filter2 は同様の処理をおこなっている。filter2 以外のプログラムは通常の CPU を想定して書かれたプログラムである。

	filter	filter2	hamming	sed	grap	egrep	gzip	gs
temporal size (A1)	15%	81.1%	93.8%	0%	0%	0%	0%	0%
(A2)	15%	81.1%	93.8%	14.7%	17.0%	62.1%	82.0%	5.7%
max speedup (A1)	1.18	5.29	16.13	1	1	1	1	1
(A2)	1.18	5.29	16.13	1.17	1.2	2.6	5.6	1.06
speedup (A1)	1.08	4.11	14.4	1.0	1.0	1.0	1.0	1.0
(A2)	1.17	5.01	14.4	1.15	1.12	1.45	2.93	1.03
gate count (A1)	5120	5920	12416	0	0	0	0	0
(A1)	5120	5920	12416	1120	5504	5984	11712	960

表 4: temporal size and program speedup

表 4 に評価結果を示す。各プログラムの temporal size を見ると、filter の 15% と比較し filter2 では 81.1%とかなり改善されていることがわかる。これは、先に述べた様にライブラリと浮動小数点演算という 2 つの制約を取り除いたことによる。単純な作業と繰り返しとなる hamming では非常に高い temporal

size を得ている。それ以外の 5 つのプログラムにおいては、アーキテクチャ A1 の設定では FPGA を用いても性能向上する部分が見つからずに、temporal size が 0 となっている。

性能向上を見てみると、もっとも高い hamming で 14.4 倍の性能向上を達成している。しかし、sed, grep, gs においてはアーキテクチャ A1 では性能向上 0、アーキテクチャ A2 においても数十%の性能向上に留まっている。

FPGA で使われたゲート数では、hamming でもっとも多い 12416 ゲートを用いている。ハードウェア・パートに分離した全ての候補の平均ゲート数は 2017、その中の 90%の候補はゲート数 4000 より少なかったとしており、それほど多くのゲート数を利用していないことがわかる。

2.2 Reconfigurable datapath architecture

文献 [2] では、動的にデータパス構成を変更するアーキテクチャとして Reconfigurable datapath architecture (rDPA) を提案している。これは、従来の CPU を新しいアーキテクチャで置き換えようとする試み

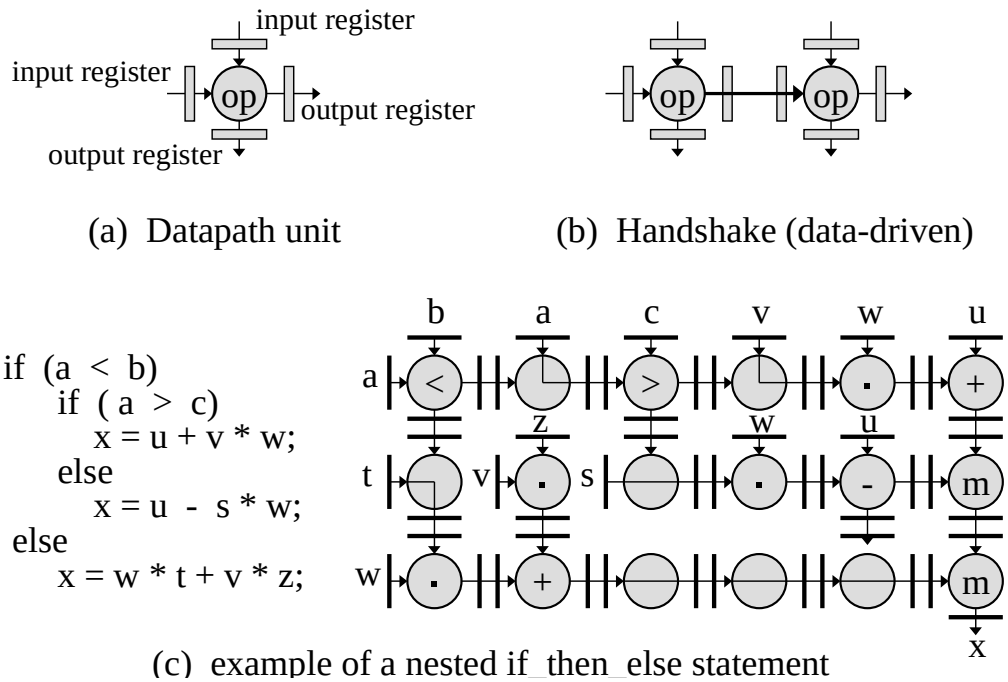


図 11: Datapath unit and Example

で、利用する IC として FPGA の利用を検討している。図 11(a) はデータパスを構成する要素で、datapath unit と呼ばれる。この datapath unit は 2 つの入力レジスタと 2 つの出力レジスタ、それらの演算をおこなう演算部分を持つ。演算の粒度は 1 ビットの演算や 1 ワードの演算などに再構成可能となっている。datapath unit は図 11(b) に示す様にデータの到着を待って即次の計算を始めるデータ・ドリブンで処理を進めていく。図 11(c) は rDPA におけるデータパス構成の例を示した。

2.3 Self-reference Evolving Architecture

演算結果がアーキテクチャの構成を変える。進化・生命の起源をシミュレート。このような事ができそうだという段階で、具体的な取り組みはない。

3 Programmable Logic Deviceの利点

PLDの構造と技術動向、PLDを用いた研究例を見てきた。ここでは、動的に構成を変更できるSRAMタイプのPLDについて、その利点をまとめる。
PLDの利点はプログラム可能であるという一言に尽きるが、さらに詳しくみていくと次のような点が利点として考えられる。

- PLDで実行する部分を実行するアプリケーションによって最適化することで、プログラム(の一部)を高速に実行することが可能となる。これは、計算機性能の向上を意味し、アプリケーションを実行するエンドユーザにとって最も基本的な要望を達成する。
- 研究におけるプロトタイプの開発や、教育における実験等において設計した回路を何回も設計し直し、デバックや改良ができる。これは、回路設計者の設計の負担を軽減し、開発に必要となるコストと期間の削減を可能とする。
- 今まで困難だった領域への挑戦。これは、進化するハードウェアの様に、今まで困難だった研究領域の実現が可能となり、画期的な進歩を期待するものである。

4 考察

従来の計算機システムにFPGAを追加したシステムにおいて最大14倍の性能向上を達成できている。
メモリ・アクセス時間が問題であり、それを解消するためにはコンパイラが最適化できるだけの情報を与えることが可能な言語を用いる必要がある。
プログラマの視点からみれば、プログラムが自動的にハードウェア・パートとソフトウェア・パートに分割され、どの部分がFPGAで実行されているか意識する必要がないが、高い性能を引き出す場合には、FPGAで最適に実行されることを意識したプログラム技術が必要となるかもしれない。

5 まとめ

容易に論理の書き換えが可能なデバイスFPGA(Field Programmable Gate Array)が注目を浴びている。従来は、値段が高い、利用できるゲート数が少ない、動作周波数が低いといった問題点から、利用は非常に制限されてきたFPGAだが、近年のデバイス技術の向上により、先の問題点の多くは改善される方向にある。
本輪講では、FPGAの構造と技術動向を解説した後に、具体的な研究例として、(1) Function Booster Ensemble (2) (3) Self-reference Evolving Architectureを紹介した。この様な計算機環境では、命令セットアーキテクチャをインタフェースとしたソフトウェアとハードウェアの分離という概念すら通用しなくなる。

参考文献

[1] Axel Jantsch, Peeter Ellervee, Johnny Oberg and Ahmed Hemani, *A Case Study on Hardware/Software Partitioning*, IEEE Workshop on FPGAs for Custom Computing Machine, pp.111-118, 1994.

[2] Reiner W. Hartenstein, Rainer Kress and Helmet Reinig, *A Reconfigurable Data-Driven ALU for Xputers*, IEEE Workshop on FPGAs for Custom Computing Machine, pp.139-146, 1994.

[3] 身次茂, *FPGA* の現状と将来, 情報処理学会 Vol.35, No.6, pp.505-510, 1994.

[4] ゲート・アレイに近付いてきた高集積/高性能 *PLD* の話題, Design Wave Maganize No. 12, pp.52-56. 1997.

[5] 越智裕之, *FPAA* : フィールドプログラマブルアキュムレータアレイ, 情報処理学会研究報告 計算機アーキテクチャ研究会, 97-ARC-126, pp.97-102, October, 1997.