情報処理学会論文誌　データベース　Vol. 4　No. 2　35–51 (July 2011)

ストリーム処理エンジン向け 動的再構成可能プロセッサアーキテクチャの設計

三好 健文[†1]　寺田 祐太[†2,*1]
川島 英之[†3]　吉永 努[†1]

動的再構成可能ストリーム処理エンジン DR-SPE のプロセッサアーキテクチャを提案する．ストリーム処理エンジンは，ときどき刻々と変化するデータの流れであるストリームデータに対して，SQL ライクな宣言的クエリ言語を用いて，関係演算や算術演算を適用できる．DR-SPE は並列処理による高い処理性能を実現すると同時に，高速なクエリ登録や演算子実行順序切換えをサポートする専用ハードウェアによるストリーム処理エンジンである．DR-SPE が提供する演算子は，Streams on Wires と同等である．本論文では，提案するアーキテクチャを FPGA XC6VLX240T-1 上に実装し，クエリの構成時間および処理性能を評価する．評価の結果は，DR-SPE は Streams on Wires と同等のスループットを実現しながら，85 μ 秒でクエリを構成できることを示す．

A Dynamic Reconfigurable Processor Architecture for Stream Processing Engine

Takefumi Miyoshi,[†1] Yuta Terada,[†2,*1]
Hideyuki Kawashima[†3] and Tsutomu Yoshinaga[†1]

A processor architecture of dynamic reconfigurable stream processing engine DR-SPE is proposed. By using declarative query language, stream processing engine is able to apply relational and arithmetic operations to stream data. DR-SPE is a special purpose hardware for stream processing, which achieves both of high processing performance by exploiting parallelism in target query and ability for query registration and execution order of operations at runtime. Available operations in DR-SPE are the same as ones in Streams on Wires. In this paper, DR-SPE is implemented on a FPGA XC6VLX240T-1, and its configuration time for operations and its performance are evaluated in real experiments. The result of experiments shows that DR-SPE realizes 85 μ second on configuration of operations, which overwhelms Streams on Wires. Simultaneously, DR-SPE achieves comparable performance with Streams on Wires,

1. はじめに

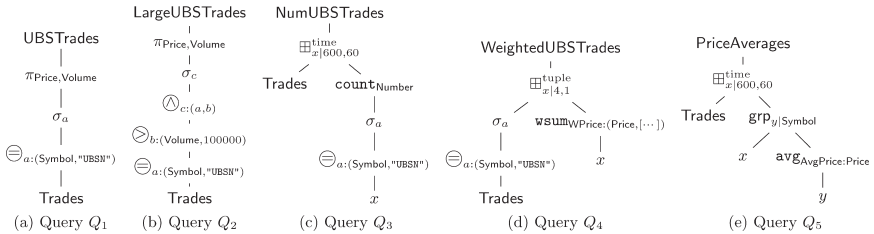
ときどき刻々と到着するデータの流れに対し，種々の演算を行うストリームデータ処理が注目を集めている．ストリームデータ処理の例には，IP パケットストリームの解析による不正アクセス検知や株価の解析などがあげられる．ストリームデータに対する解析処理を簡単に扱うことを目的として，ストリームデータ処理，ならびにそれを実現するミドルウェアであるストリーム処理エンジン（SPE）が研究されてきた．多くの SPE ではストリームデータに対して SQL ライクな宣言的言語でクエリを記述できる．そのため，SQL で扱われてきた集合に対する演算をユーザが簡便に記述できる．

SPE では連続的に到着するデータに対して，とりこぼしなくクエリ処理を実行することが求められる．ルータにおける IP パケットのようにデータ到着頻度が高い場合，これを処理するために求められる計算能力は高くなる．

高い計算能力を提供する手段として，専用ハードウェアの利用は有力な選択肢である．専用ハードウェアでは，内部に搭載可能な多数の演算ユニットの並列動作により高い処理性能を得られる．Streams on Wires[1] とそのコンパイラである Glacier[2] では，FPGA を用いたストリームデータ処理専用のハードウェアを提案している．Streams on Wires では，ストリームデータ処理を行うための基本要素をハードウェア記述言語（HDL）ライブラリとして用意し，Glacier がクエリに応じてそれらを組み合わせることで，そのクエリ処理専用ハードウェアの HDL コードを生成する．

HDL から FPGA 上で実行可能な回路情報を構成するには，合成と配置配線が必要である．2.80 GHz で動作する Core™ i7 860，メモリ搭載量 8 GiB の Linux（Fedora 13）計

†1 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†2 電気通信大学電気通信学部
Faculty of Electro-Communications, The University of Electro-Communications
†3 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
*1 現在，株式会社アバールデータ
Presently with Avaldata Corporation



(a) Query Q_1 (b) Query Q_2 (c) Query Q_3 (d) Query Q_4 (e) Query Q_5

図 1　Streams on Wires に例としてあげられているクエリ（文献 1) より引用）
Fig. 1　Sample queries listed in Streams on Wires as example (Cited from 1)).

表 1　各クエリの合成と配置配線にかかる時間
Table 1　Time for synthesis and place & route for each of sample queries.

クエリ	コンパイル時間（秒）
Q1	171.18
Q2	174.17
Q3	181.33
Q4	177.07
Q5	179.75

算機上で，Xilinx ISE 12.1 を用いて**図 1** に示す Q1 から Q5 のクエリの合成および配置配線をしたとき，**表 1** に示す時間（約 3 分）を我々の実験環境では要した[⋆1]．さらに，JTAG を用いる FPGA への構成情報のダウンロードには，約 1 分ほどを必要とする．したがって Streams on Wires の方式では，頻繁な新規クエリの登録や動的クエリ最適化の実現は困難である．

本論文の貢献は，独自の再構成機構を有する動的再構成可能なストリーム処理エンジン（DR-SPE）のプロセッサアーキテクチャを提案し，そのハードウェアコスト，信号遅延，および再構成に必要な時間を評価することである．提案するプロセッサアーキテクチャでは，SPE を構成するビルディングブロックをあらかじめハードウェア上に構築し，この方式では，クエリの構成時間が数十〜数百 μ 秒程度まで短縮されるため，頻繁な新規クエリの登録や動的クエリ最適化の適用が可能になる．

DR-SPE の設計においては以下の課題が現れる．まず，想定する処理内容をカバーするための単位演算ユニットやデータ授受を効率的に実行するためのユニット間接続機構などの演算構成要素が求められる．次に，再構成を可能にするために必要となる使用リソース量や信号遅延の増加を最小化するという，容易ではない実装方式が求められる．

本論文では DR-SPE の実現を目指し，プロセッサアーキテクチャ設計の初期検討を行う．まず，DR-SPE を構成する演算構成要素について検討する．さらに，検討した演算構成要素により，Streams on Wires の Algebra を実現できることを示す．また，提案するプロセッサアーキテクチャを FPGA 上に実装した場合の回路規模および信号遅延を示す．

本論文の構成は次のとおりである．2 章で本研究の対象であるストリームデータ処理について述べる．3 章では，DR-SPE の設計課題と提案プロセッサアーキテクチャを述べ，提案プロセッサアーキテクチャが Streams on Wires と同等の処理機能を有することを示す．4 章では，提案する DR-SPE に所望のクエリを設定するために必要なデータサイズを評価し，構成時間を見積もる．さらに，動的再構成機構を含む DR-SPE のハードウェアリソース量および信号遅延に関するオーバヘッドを評価する．最後に 5 章で，まとめと今後の課題を述べる．

2. ストリームデータ処理

各種センサデータ，Twitter，Ustream，実時間株価情報配信，移動体位置データ，監視カメラなど，実世界の状況を連続的に配信する情報源が多数出現している．このような情報源により配信されるデータをストリームデータと表記する．ストリームデータは頻繁に生成される．東京証券取引所では 2 ms 程度で株価を生成し[3]，ルータは 300 Tbps 程度でデータを生成し[4]，一般的な産業用ロボットは 1 ms 周期でモータ制御に関するデータを生成する．一方，データ生成レートは低くても，データ生成総量が大きい場合もある．たとえば，カメラは 1 秒間に 30 枚程度の画像データを生成するのみだが，監視カメラの数は増加の一途をたどっている．

ストリームデータ処理において最も重要な研究課題の 1 つに，大規模ストリームデータに対する高性能化がある．まず，先進的なハードウェアを積極的に活用することで高性能化を目指すアプローチとして，ネットワークプロセッサを用いる手法[5]，GPGPU を活用した GPUTeraSort [6]，Cell/B.E. を用いたストリームに対する Join の高速化[7] の研究および，FPGA を用いたストリームデータ処理ハードウェアを実現する手法の研究[1),8),9)] がある．ここで，本論文で対象とする上述のストリームデータに対する処理では，システムのボトルネックは，個々の計算負荷ではなく，I/O やメモリバンド幅にある．したがって，GPGPU

⋆1 複雑な回路の場合には合成時間が増加する．たとえば，囲碁専用ハードウェアなどでは，3 時間以上のコンパイル時間が必要な場合がある．

や Cell/B.E.，ネットワークプロッサ上のソフトウェアによる処理ではなく，I/O アクセスポートやメモリバンド幅を自由に設計する専用アーキテクチャの設計は有効なアプローチである．

データベース処理に特化した専用計算機システムとしては，DIRECT [10] や文献 11) にあげられるシステムがある．近年では，開発および製造コストの点から，専用アーキテクチャを ASIC として独自に生産するのではなく FPGA を用いて実現する手法が好まれている．Netezza Corp. では，データベース処理に特化したハードウェアシステムとして，組み込みディスクドライブ，CPU，メモリと FPGA からなる SPU を多数有する Netezza Performance Server® を設計，市販している．SPU では FPGA の用途をデータのフィルタリングに活用しているが，Streams on Wires [1] では，FPGA 上にユーザの所望するクエリを柔軟に実現する手法を提案している．

一方で，アルゴリズム的な観点から高性能化を実現するために重要な役割を果たす技術として，動的クエリ最適化[12),13)] がある．これは，選択演算，結合演算，射影演算などの関係演算の実行順序を状況に応じて動的に変更することで，処理性能を向上させる技術である．タプル到着にともなって，結合率や選択率を高速に推定し，それに従って演算子実行順序を入れ替えることで，大幅な性能改善を達成できる可能性がある．ただし，演算子実行順序の入替え処理には高速性が求められる．たとえばミリ秒単位で変動する証券取引所のデータを対象とすれば，入替え処理は遅くとも 1 ミリ秒以下である必要がある．

前述の FPGA 上に実装した専用ハードウェアによってストリーム処理を高速化する既存研究[1),8),9)] では，次章で述べる理由から，これを実現することが困難である．そこで本研究では，高速入替え処理を実現可能な，ストリーム処理専用ハードウェアの設計に挑む．

3. 動的再構成可能ストリーム処理エンジン

ストリームデータ処理を専用ハードウェア化すれば，サイクルレベルでの処理と並列処理により，処理性能を向上させることができる．Streams on Wires [1] では，FPGA を用いてストリームデータ処理を行う手法を提案している．FPGA は，回路を構成する構成情報によって，実行前にその処理内容を変更できる再構成可能なハードウェアデバイスである．Streams on Wires では所望の FPGA 上にストリーム処理エンジンを構築するためのビルディングブロックとして**表 2** に示す Algebra と表記される演算要素を定義している．各 Algebra は**図 2** に示すように，出力にレジスタを持つ回路ユニットとして統一したインタフェースの下で定義されている．そのため，クエリを構成するために必要となる Algebra を自由に組み合わせることができる．構成された FPGA 上の回路はパイプライン並列性によって高い処理性能を発揮する．

表 2　Streams on Wires において定義される演算要素である Algebra（文献 1) より引用）
Table 2　Algebra defined as operation elements in Streams on Wires (Cited from 1)).

(1) Projection	$\pi_{a_1,\ldots,a_n}(q)$	projection
(2) Selection	$\sigma_a(q)$	select tuples where field a contains true
(3) 算術/論理演算	$\circledast_{a:(b_1,b_2)}(q)$	arithmetic/Boolean operation $a = b_1 \star b_2$
(4) Union	$q_1 \cup q_2$	union
(5) Aggregation	$agg_{b:a}(q)$	aggregate agg using input field a, $agg \in \{\mathbf{avg}, \mathbf{count}, \mathbf{max}, \mathbf{min}, \mathbf{sum}\}$
(6) Grouping	$q_1\ \mathbf{grp}_{x\|c}\ q_2(x)$	group output of q_1 by field c, then invoke q_2 with x substituted by the group
(7) Windowing	$q_1\ \boxplus^t_{x\|k,l}\ q_2(x)$	sliding window with size k, advance by l; apply q_2 with x substituted on each wind.; $t \in \{\text{time}, \text{tuple}\}$: time-, or tuple-based
(8) Concatenation	$q_1 \infty q_2$	concatenation; position-based field join

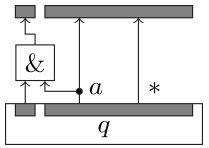


図 2　ビルディングブロックとしての Algebra 回路（文献 1) より引用）
Fig. 2　Algebra circuit as building block of query plan (Cited from 1)).

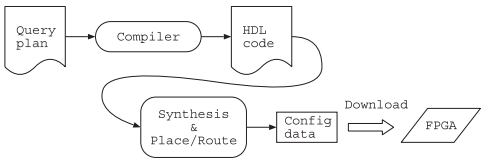


図 3　Streams on Wires [1] 処理におけるエンジン生成フロー
Fig. 3　A flow of generating SPE in Streams on Wires [1].

図 3 に，FPGA 上でのストリームデータ処理の生成フローを示す．Streams on Wires では，クエリをコンパイラ（Glacier）によってコンパイルし，必要となる Algebra のインスタンシエーションおよび Algebra 間の配線を定義する HDL コードを生成する．生成され

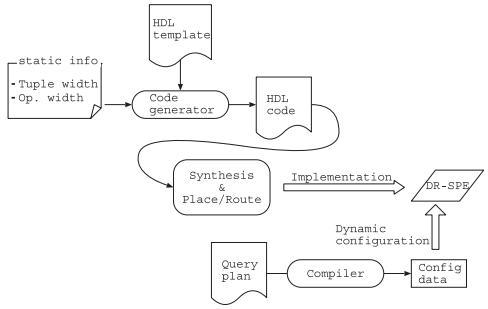


図 4　動的再構成可能ストリーム処理エンジン（DR-SPE）の生成フローとクエリの設定
Fig. 4　A flow of generating dynamic reconfigurable stream processing engine (DR-SPE) and configuration of query plan for it.

た HDL コードを，既存の FPGA 開発ツールによって合成，配置配線することで，対象とする FPGA 上の構成情報が得られる．

この方式では，新しいクエリを登録する場合に，FPGA 開発環境が必要であるし，しかも合成と配置配線に数分の時間がかかる．一方，新規クエリの登録や，流れてくるデータに応じて演算子の順序を入れ替える動的クエリ最適化には遅くともミリ秒単位での実行が求められる．したがって，この方式を用いた短時間での新しいクエリの登録や動的問合せ最適化の処理の実現は困難である．

そこで本論文では，HDL からの合成と配置配線を行わずにクエリの変更や追加，最適化ができる動的再構成可能ストリーム処理エンジン（DR-SPE）を提案する．DR-SPE 上でクエリを実行するまでのフローを**図 4** に示す．図 4 のフローでは，FPGA 上に直接クエリを構成する図 3 のフローと違い，再構成可能なプロセッサアーキテクチャとして設計された DR-SPE の内部のレジスタの設定を動的に書き換えるパス “Dynamic configuration” により，クエリを実現する．図 4 のフローでは，まず，対象とするアプリケーションに応じて，タプル幅や演算幅などの設定情報によりカスタマイズした DR-SPE の HDL コードを生成する．この HDL コードは合成，配置配線され，ハードウェア回路として得られる．この時点では，特定のクエリが設定されるわけではない．クエリは，別途，専用のコンパイラによって DR-SPE 用の構成情報にコンパイルされる．構成情報は，実行時に，“Dynamic configuration” の経路で DR-SPE に転送される．

動的再構成可能ハードウェアである DR-SPE は次の特徴を持つ．

- ストリームデータ処理を実行可能
- サイクルレベルでの処理と並列性の活用により高い演算性能を実現
- 内部モジュールの動作の処理内容を処理中に即時に変更可能
- 処理内容の部分的な追加や変更，ならびにパラメタの設定が可能

この特徴により DR-SPE は汎用プロセッサ上のソフトウェアに比べて高い処理性能を達成する．この性質はさらに，動的なクエリの追加や変更ができるという ASIC による専用ハードウェアにはない柔軟性を達成する．

DR-SPE は，クエリを構成するために必要な共通の演算構成要素で構成される．クエリコンパイラは，それらの構成要素を用いてクエリを実現するための構成情報を生成する．

FPGA 上で所望の処理を実現するためには，処理を実現するための論理回路を生成し，その回路を，FPGA を構成するルックアップテーブル（LUT）上に真理値表としてマッピングしなければならない．一方で，DR-SPE の構成要素はデータ処理の機能レベルで定義されているため，FPGA を構成する LUT に比べて粒度が大きい．そのため，構成情報を生成するための計算コストおよび構成情報のデータサイズが大幅に削減される．さらに，一般に FPGA 上の回路は，そのすべてを書き変える必要があるが，DR-SPE 上のクエリを変更する場合には，変更に関する特定の構成要素の設定のみを更新すればよい．これらの，構成情報のサイズが小さいことおよび部分的な再構成が可能であることによって，Streams on Wire に比べ，DR-SPE に対する構成情報の生成に関する時間および，構成情報を設定するための時間は短縮される．

3.1　DR-SPE の設計課題

本節ではストリーム処理を実現可能な動的再構成可能な DR-SPE を実現するために必要な機能および，設計課題について述べる．

所望の処理を複数の基本的な演算器を組み合わせることで実現する動的再構成可能プロセッサの従来研究には，ADRES [14] や FE-GA [15] などがある．FE-GA や ADRES は信号処理を高速化するためのアクセラレータとして設計されている．すなわち FE-GA や ADRES は，信号処理に多用される加減算や乗算の組合せを，データフローのまま演算器群に割り当てることで，細粒度データ並列性とパイプライン並列性を活用した高速化を実現する．

一方，ストリームデータ処理は，Union，Windowing，および Grouping といった，複数のデータパスから特定のデータを選択する処理や，指定条件下で送受信の可否を決定する処理など，データフローへの展開が自明でない演算を数多く含む．これらの演算を実現するためには，データフローがマッピングできるだけではなく，データパスの切換え，およびスト

リームデータの入出力を制御する機構が必要になる．

この機構は FE-GA や ADRES には存在しない．DR-SPE はストリームの入出力を制御する機構として，複数の演算器の入出力の組合せを実行時に自動的に切換可能な機構を有するスイッチボックス（3.2.2 項）と，送受信の可否を能動的に制御できるストリーム入出力制御器（3.2.3 項）を備える．これらの機構により DR-SPE はストリームデータ処理を実現する．

ここで注意されるべきは，データパスの切換えがマルチプレクサを基本要素として実現されることである．マルチプレクサは複数の AND 演算から構成される．したがって大規模な切換えを行うマルチプレクサを実現するためには，多数の AND 演算が必要になる．それらの実現には多量のハードウェアリソースを必要とする．また，AND 演算通過数が増加するから信号遅延が増加する．信号遅延の増加により，組合せ回路の出力が確定するまでの時間が長くなるため，正しく回路を動作させるためには，全組合せ回路の同期をとるクロックの周期，すなわち動作周波数を低く設定しなければならない．これは，処理スループットの低下および，処理遅延増加の原因となる．DR-SPE 上で頻繁にクエリの再構成を行う場合の処理スループットの上限は，DR-SPE の再構成機構が提供しうる再構成時間に抑えられる．そのためマルチプレクサの規模による数十 n 秒オーダの遅延増加は処理スループットに影響しないとも考えられる．しかしながら，この遅延増加は，再構成を行わない間の処理スループットおよび，信号遅延に影響する．特に，大規模なクエリの実行においては，遅延が加算されるため，大規模な信号遅延の増加，すなわちレスポンスの悪化につながる．

DR-SPE の設計課題は，再構成ハードウェア機構により増加するハードウェアリソース量の抑制，同機構により増加する信号遅延の抑制，および再構成時間の最小化，の 3 点である．これらを下記にまとめる．

設計課題 1：ハードウェアリソース量の抑制　第 1 に，再構成ハードウェア機構により増加するハードウェアリソース量の抑制問題について述べる．仮にすべての要素が積算や除算などの複雑な（処理速度向上のために複数サイクルに分割して実行されるべき）演算を実現できるように設計すると，比較演算である == や > など単一サイクルで実行可能な単純な演算を行う場合のハードウェアリソースに対する使用率が低下し，無駄が大きくなる．また，サイクルの異なる処理のパス間でデータを同期させるために，処理と同じだけの任意のサイクル数の遅延を挿入する必要があり，そのためのハードウェアリソース量増加のオーバヘッドも無視できない．

設計課題 2：信号遅延の抑制　第 2 に，再構成ハードウェア機構により増加する信号遅延の抑制問題について述べる．これはデータ授受にかかる配線の自由度の問題ともいえる．クエリから HDL コードを生成する場合には，所望の回路間を自由に接続することができ，また，処理過程で生じる中間結果を受け渡すための信号を簡単に追加できる．一方，ハードウェアにおいては配線は静的に決定されており，自由な接続相手の選択や，データバス幅の変更はできない．動的に選択可能な接続相手の候補を増やすためには，大きなマルチプレクサが必要になり，信号遅延が増加する．したがって，増加する信号遅延を抑制しながら，接続関係を柔軟に選択できるプロセッサアーキテクチャ設計が必要である．また，回路合成後（動的再構成段階）において任意の信号を追加するには，配線そのものを回路として実装する必要がある．これは信号遅延およびハードウェアリソース量に鑑みれば，実現不可能であると考えられる．

設計課題 3：再構成時間の最小化　第 3 に，再構成時間の最小化について述べる．選択可能なパラメータが多い場合には設定に必要な情報量は多くなり，転送時間と設定時間が増加する．不要なパラメータや選択肢，命令を削減することで，再構成に必要な情報量を小さくし，再構成にかかる時間を短くすることが課題となる．

3.2　提案プロセッサアーキテクチャ

3.1 節に述べた 3 点の課題を解決するために，DR-SPE のプロセッサアーキテクチャを次の制約および設計目標に基づいて設計する．

- 各構成要素の処理は，単一サイクルで終了するものに限定する．
- データバス幅は固定ビット幅とする．
- 設定に必要なレジスタ数を削減するため，設定項目を削減する．

Streams on Wires で定義されている Algebra と同等の機能を DR-SPE において実現するには，算術/論理演算や Aggregation を実現するための演算機構および演算機構どうしのデータ授受のパスが必要になる．また，Union や Grouping，そして Windowing を実現するためには，複数の入出力を条件に従って選択し，制限する機構が必要となる．すなわち，所望のクエリに合わせた処理を実現するためには，これらの機構とパスをビルディングブロックとして用意し，それらのビルディングブロックに対する設定を与えられるようなプロセッサアーキテクチャを設計しなければならない．

図 5 に，提案する DR-SPE のプロセッサアーキテクチャの概観を示す．DR-SPE はタイル状に配置された単位演算ユニット（Operation Unit）で構成される．単位演算ユニットどうしは，隣接するユニットの接続関係を決定するスイッチボックス（Switch Box）によって接続される．複数の単位演算ユニットは，単位演算ユニットブロックにまとめられる．ス

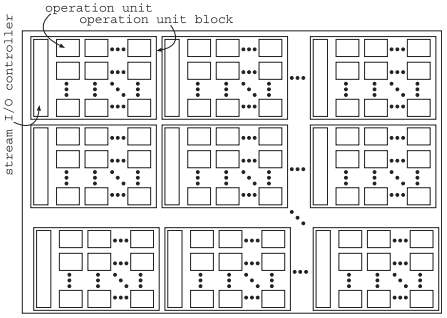


図 5　DR-SPE プロセッサアーキテクチャ概要
Fig. 5　An overview of DR-SPE architecture.

トリーム入出力制御器（eis/eos）は，各ブロックに 1 つずつある．各単位演算ユニットブロック内では，ストリームの入出力は統括的に制御される．すなわち，同じブロック内では，あるユニットの出力を許可し，あるユニットの出力を禁止するといった，連携したストリーム入出力処理ができる．

これら，単位演算ユニット，スイッチボックス，ストリーム入出力制御ユニットは，それぞれの動作を決定するための設定レジスタを持つ．そのレジスタの値を，図 4 の “Dynamic configuration” のパスで書き換える仕組みを実装する．この仕組みによって，DR-SPE の動作の動的な変更が実現される．

提案するプロセッサアーキテクチャにおける，3.1 節に述べた 3 点の課題の解決手法について，以下に述べる．

設計課題 1 に対する解決手法　設計課題 1 に示したハードウェアリソース量の抑制を実現するために，DR-SPE を構成する構成要素のデータバス幅は，入力されるストリームデータと演算結果を保持するフィールドによる，固定幅とする．複数の値をユニット間で共有する必要がある場合には，コンパイラが固定されたフィールド内での使用位置をスケジューリングする．このことにより，配線を柔軟に変更できるハードウェア機構を簡略化できる．

設計課題 2 に対する解決手法　設計課題 2 に示した信号遅延の抑制を実現するために，提案プロセッサアーキテクチャを構成する単位演算ユニットとスイッチボックス，ストリーム入出力制御器中の組合せ回路の信号遅延が同程度となるように設計する．このことにより，ハードウェア回路中の信号遅延を小さく抑え，最高動作周波数を高く保てる．各単位演算ユニットは単純な演算機能のみを提供するが，複雑な演算処理は，複数の構成要素を組み合わせることで実現できる．所望の処理を単一サイクルの処理に分割する場合，処理に必要なクロックレイテンシは増加するが，処理は必ず 1 サイクルで完了するため，処理スループットは 1 tuple/サイクルを保証できる．また，単位演算ユニットの機能を単純なものに限定することで，DR-SPE 上に構成されるクエリのハードウェアリソース使用率を高くでき，設計課題 1 に示したハードウェアリソース量の抑制にも有効である．

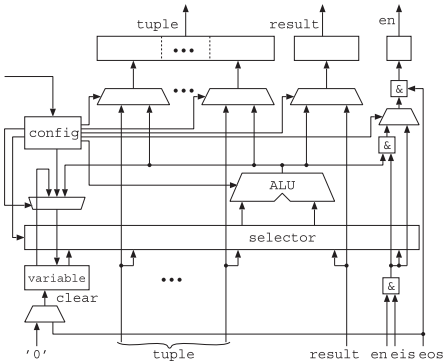


図 6　単位演算ユニット
Fig. 6　Operation unit.

設計課題 3 に対する解決手法　単位演算ユニットで実現可能な演算を単純なものに制限すること，データバス幅が固定であることにより，DR-SPE に設定しなければならない構成内容を表現するデータサイズが小さくなる．したがって，処理の実現に必要なパラメータが少なくなり，設計課題 3 に示した再構成時間の最小化が達成できる．

本節では，まず，単位演算ユニット，スイッチボックスおよびストリーム入出力制御器についてそれぞれ説明する．これら 3 つの機能ユニットで実現される機能には重複がない．また，定義した構成要素を組み合わせることで Streams on Wires の Algebra と同等の演算機能を実現できることを示す．

3.2.1　単位演算ユニット

図 6 に単位演算ユニット（Operation Unit）のプロセッサアーキテクチャを示す．各単位演算ユニットの入出力は，処理の対象となるタプル（tuple）と演算結果を格納するフィールド（result），および，その信号が有効であることを示すフラグ（en）で構成される．result

表 3　単位演算ユニットで実現可能な演算
Table 3　Available operations by each operation unit.

種別	演算
算術演算	和，差，+1，−1
シフト	左 1 bit シフト（ローテート有/無）， 右 1 bit シフト（ローテート有/無）
論理演算	論理積，論理和，排他的論理和，否定
比較	==，>，>=，!=

は固定長のビット列として，この単位演算ユニットが合成される時点で決定される．タプルは対象データに対して固定長で与えられる．すなわち，単位演算ユニットの入出力のデータ幅は固定である．

各単位演算ユニットは，基本的な演算を行うための ALU を 1 つ持つ．この ALU で実行可能な演算を**表 3** に示す．これらの演算を実装する回路の信号遅延は十分小さい．したがって，すべての単位演算ユニットの処理を 1 サイクルで完了できる．

ALU への入力は，selector によって，ALU の演算 bit 長で分割したタプル上の指定した bit 列，result，en および内部変数 variable の候補から選択される．演算結果は，タプル，result，en，および variable のいずれにも出力可能である．各フィールドへの出力は演算結果と入力値とのマルチプレクサによって選択できる．

また，単位演算ユニットに対する入出力を，強制的に制御可能な eis（enable input stream）および eos（enable output stream）入力を備える．eis および eos が ‘0’ [⋆1]のとき，強制的に入力/出力信号の en が ‘0’ に設定され，データは無効化される．

3.2.2　スイッチボックス

スイッチボックスは，3.2.1 項で述べた単位演算ユニットどうしの接続関係を保持する．スイッチボックスを介して，ある単位演算ユニットの入力は，その 7 方向[⋆2]に接続された単位演算ユニットの出力のいずれかを受けることができる（**図 7**）．DR-SPE のスイッチボックスでは，入力を受け取る方向を，(1) 静的に 1 つの入力に固定，(2) 指定した 2 方向からの入力を切り換え，の 2 種類の方法で設定できる．(2) の場合，カウンタによって指定されたタイミングにより，選択される入力が自動的に切り換わる．

スイッチボックスのブロック図を**図 8** に示す．valueA，valueB，selA，selB の値は実行時

⋆1 ‘0’ および ‘1’ は 1 bit の信号がそれぞれ 0 あるいは 1 であることを表す．
⋆2 各種クエリ処理を行うために十分な数であり，「接続しない」を入れても 3 bit で定義できるため．

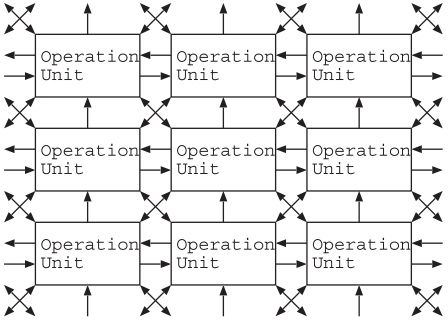


図 7　スイッチボックスによる単位演算ユニットの接続
Fig. 7　Connection between operation units by swtitchbox.

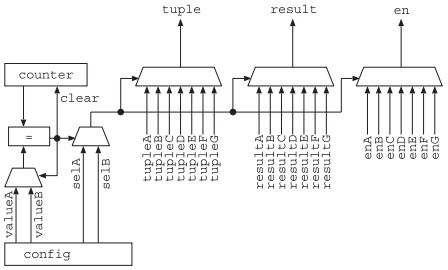


図 8　スイッチボックスのアーキテクチャ
Fig. 8　A block diagram of switchbox architecture.

に設定される．selA および selB が出力対象とすべき入力ポートの選択に相当する．valueA および valueB は，selA と selB の設定を有効にする期間を設定する値である．カウンタが valueA（あるいは valueB）とマッチすると，出力するポートは selA で指定したポートから selB で指定したポート（あるいは selB で指定したポートから selA で指定したポート）へ切り換えられる．

3.2.3　ストリーム入出力制御器

単位演算ユニットにより所望の処理がデータに対して適用でき，スイッチボックスにより単位演算ユニットどうしのデータ授受関係を自由に決定できることを述べた．一方，Aggregation では入力データをあるタイミングまで加算するなどという処理が必要である．この処理を実現するためには，他の単位演算ユニットの結果によって，カウンタ動作を行うユ

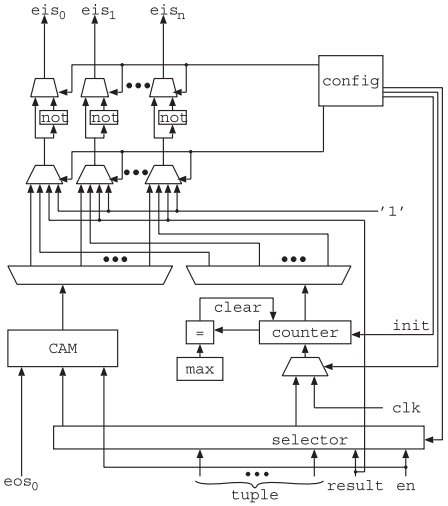


図 9　ストリーム入力制御器のアーキテクチャ
Fig. 9　A block diagram of stream input controller.

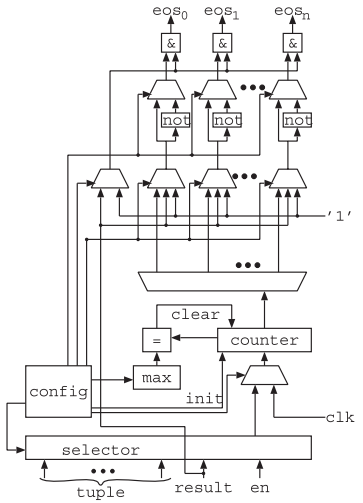


図 10　ストリーム出力制御器のアーキテクチャ
Fig. 10　A block diagram of stream output controller.

ニットのデータ出力を制御する必要がある．また，Windowing および Grouping を実現するためには，単に個々のデータを処理するだけではなく，複数の演算結果からの出力の選択や，不要なデータの入力を制限するといった複数の単位演算ユニットを連携させる処理が必要になる．

入出力を制限する機能は，単位演算ユニットの eis と eos によって提供される．したがって，eis あるいは eos を外部から制御できる機構があればよい．提案プロセッサアーキテクチャでは，この仕組みをストリーム入出力制御器で提供する．**図 9** および **図 10** にストリーム入出力制御器を示す．これらは，1 つの単位演算ユニットブロックに属する単位演算ユニット群の eis および eos を制御する．ブロック内の単位演算ユニットはそれぞれを識別するためのインデックスが付与されている．

eis 制御の入力は，カウンタのデマルチプレクサ（DEMUX）出力，連想メモリ（CAM）の出力のデマルチプレクサ出力，入力データの result あるいは ‘1’ の 4 候補のいずれかになる．ただし，カウンタの最大値に設定した値以上のインデックスが付与されている単位演算ユニットへの eis は必ず ‘1’ に設定される．つまり，最大値が 0 の場合には，入力データの有効/無効フラグは，つねに入力データ中の en に従う．eis への出力結果は反転，非反転信号のいずれかを選択できる．CAM は LUT を利用して実装され非同期で読み出すことができるが，実装にかかるコストが大きいため DR-SPE 中に実装可能な個数は制限される．

eos 制御の入力には，カウンタのデマルチプレクサ出力，入力データの result および ‘1’ の 3 候補のいずれかを選ぶことができる．eis の制御と同様に，カウンタの最大値に設定した値以上のインデックスが付与された単位演算ユニットへの eos は必ず ‘1’ に設定される．eos への出力結果は反転，非反転信号のいずれかを選択できる．

3.3　Algebra の実現可能性

DR-SPE の構成要素である単位演算ユニット，スイッチボックス，ストリーム入出力器を適切に設定することで，Streams on Wires の Algebra 相当の処理を実現できる．以下，各演算を実現する方式を，その設定方法を示しながら説明する．

3.3.1　Selection（$\pi_{a_1,\ldots,a_n}(q)$）と Projection（$\sigma_a(q)$）

タプル中の特定の bit 位置の値（‘1’ または ‘0’）あるいは result の値と適切な定数値の論理積の結果と，およびその結果と en の論理積で，Selection を実現できる．**図 11** に，result の 0 bit 目を対象とした Selection を実現する場合の単位演算ユニットの設定例を示す．selector は variable（定数値 1 が設定されている）と result を ALU に入力する．タプ

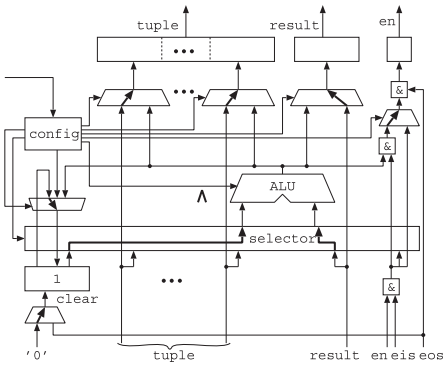


図 11　Selection の実現方式
Fig. 11　A configuration of operation unit for Selection.

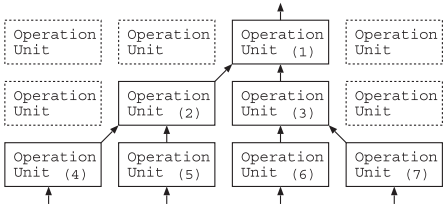


図 12　4-way Union の実現方式
Fig. 12　A configuration of switchbox for 4-way Union.

ルおよび result の出力段の MUX では，それぞれ入力された値を選択し，en の MUX では ALU における論理積結果と入力データの en の論理積の出力を選択している．Projection は，単に対象となる位置のデータを無視することで実現できる．

3.3.2　算術/論理演算（$\circledast_{a:(b_1,b_2)}(q)$）

算術/論理演算は単に ALU に実行したい演算を設定し，対象とする入力をタプルのデータ位置あるいは result から選択すればよい．出力は result または variable に書き出される．

3.3.3　Union（$q_1 \cup q_2$）

スイッチボックスのデータソースを適切に切り換えることで Union を実現できる．Union の対象となるデータを出力する単位演算ユニット群をスイッチボックスを介して接続すればよい．

図 12 に 4-way の Union を実現するためのスイッチボックスの設定を示す．(2) と (3)

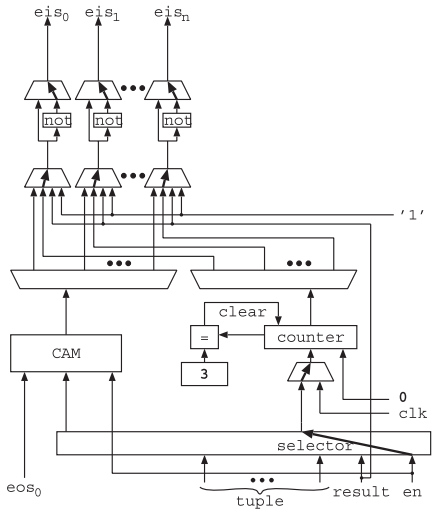


図 13　Windowing の実現方式（ストリーム入力制御器の設定）
Fig. 13　A configuration of stream input controller for Windowing.

の出力を (1) に入力するスイッチボックスでは，入力元の方向を決定する selA と selB に (2) と (3) を設定し，切換えのタイミングを決定する valueA と valueB に定数 1 を設定する．こうすることで，(1) に対して (2) あるいは (3) を出力するスイッチボックスでは，(2)(2)(3)(3)(2)(2)... と 2 サイクルに 1 度入力元が切り換わる．

同様に，(2) の入力 (4) と (5) および，(3) の入力 (6) と (7) は，それぞれ 1 サイクルずつ入力元の選択が切り換えられ，(4)(5)(4)(5)(4)...，あるいは (6)(7)(6)(7)(6)... と選択される．結果的に (1) に与えられる入力には，(4)(5)(6)(7)(4)... と各入力元からラウンドロビンで選択される．

実現可能な Union のウェイ数はスイッチボックスの入力ソースを切り替えるためのカウンタの bit 幅に依存する．カウンタの bit 幅が大きいほど動的再構成時の構成可能パターン数が向上するが，一方で回路規模が増加する．この値は DR-SPE の HDL コード生成時にカスタマイズされる．

3.3.4　Windowing（$q_1 \boxplus^{t}_{x|k,l} q_2(x)$）

ストリーム入出力制御器のカウンタを使用してサイクリックに各 eis および eos を設定することで所望の処理を実現できる．**図 13** および **図 14** にサンプルとなる Windowing の実

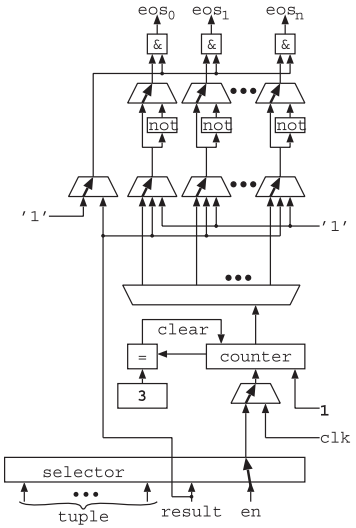


図 14　Windowing の実現方式（ストリーム出力制御器の設定）
Fig. 14　A configuration of stream output controller for Windowing.

現方式を示す．

図 13 および図 14 では，有効なデータが到達するたびにサイクリックに値の入力と出力をそれぞれ有効にする．入力は特定の 1 つの単位演算ユニットへの入力のみを無効にするため，eis の出力は最終段で反転する．また，出力のデコーダの結果を入力のデコーダの結果から 1 ずらすために，eos を制御するカウンタの初期値を 1 に設定している．

3.3.5　Aggregation（$agg_{b:a}(q)$）

単位演算ユニットでは，ALU の入出力にかかるマルチプレクサを適切に設定することで，内部に持つレジスタ variable に途中演算結果を格納することができる．variable に格納されている値と次の入力データに演算を適用することで，Aggregation の機能が実現できる．結果を出力するときには eos を ‘1’ に設定すればよい．このとき，variable の中身もクリアされる．

図 15 に Aggregation の例として有効データ（en が ‘1’ のタプル）のカウンタを実現するための設定を示す．ここでは，en を値 ‘1’ と ‘0’ をそれぞれ 1 あるいは 0 として variable に加算する．加算した結果は eos に ‘1’ が与えられるタイミングで出力される．このとき，

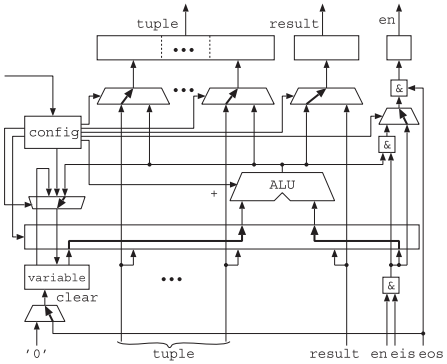


図 15　有効データをカウントする Aggregation の実現方式
Fig. 15　A configuration of operation unit for Aggregation.

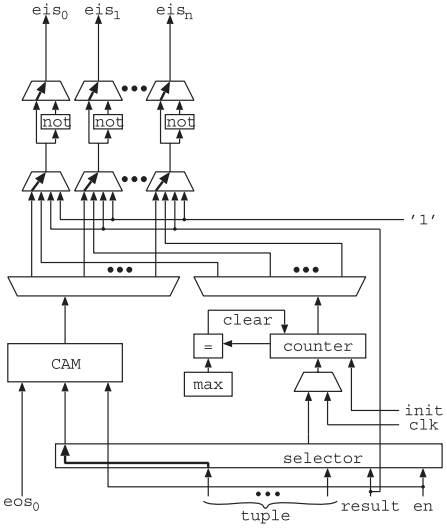


図 16　Grouping の実現方式（ストリーム入力制御器の設定）
Fig. 16　A configuration of stream input controller for Grouping.

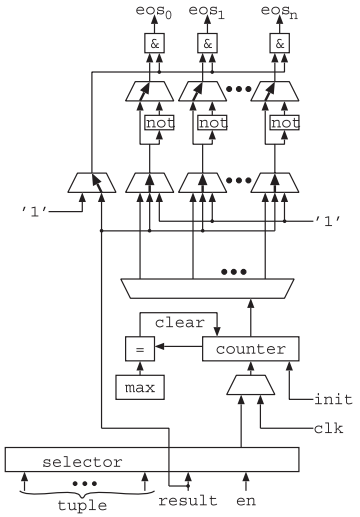


図 17　Grouping の実現方式（ストリーム出力制御器の設定）
Fig. 17　A configuration of stream output controller for Grouping.

variable の内容は 0 にクリアされる．

Aggregation である max，min，count，sum は，単一の単位演算ユニットを適切に設定することで実現される．また，avg は，2 のべき乗数で指定された固定の個数の場合であれば，sum を行う単位演算ユニットとシフト演算を行う単位演算ユニットを組み合わせることで実現される．任意の個数のデータに avg を適用するためには，除算を構成するために複雑な組合せが必要となる．

3.3.6　Grouping（$q_1\ \mathbf{grp}_{x|c}\ q_2(x)$）

Grouping はストリーム入出力制御器を活用することで実現できる．**図 16** に Grouping を実現するためのストリーム入力制御器の設定を示す．CAM を使用することでデータを単位演算ユニットに振り分ける機能を実現する．CAM のエントリは，eos の出力によってクリアされる．

eos を制御するストリーム出力制御器の設定を**図 17** に示す．ここでは単位演算ユニットの計算結果に従って eos の値を決定している．Grouping の eos は，たとえば，図 1 の Q5 のクエリのように Windowing その他の eos と連動して制御されることが多いと考えられる．そのような場合，同じ入力をストリーム出力制御器の入力に供給することで，連動した制御が実現できる．

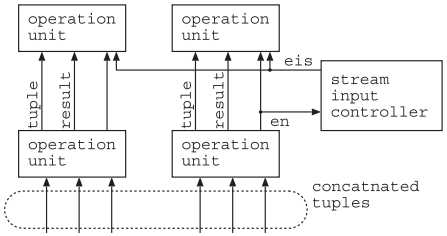


図 18　Concatenation の実現方式
Fig. 18　A configuration for Concatenation.

3.3.7　Concatenation（$q_1 \infty q_2$）

Concatenation を実現するためには，複数の単位演算ユニットをひとまとまりとしてコンパイラで扱えばよい．ここで，Concatenation の対象となったいずれかのタプルの有効/無効フラグ（en）を変更する場合には，ひとまとまりとして取り扱うすべてのタプルの有効/無効を連動して変更しなければならない．

この機能は，ストリーム出力制御器を用いることで実現できる．**図 18** に 2 つの入力タプルを Concatenation する処理の実現方法を示す．図 18 右の単位演算ユニットの処理結果によって，en の値が変化するとき，この値を，ストリーム入力制御器に入力し，得られる eis 出力を Concatenation の対象となる次段の単位演算ユニットのすべてに入力する．これにより，Concatenation の対象となった両方のタプルに一貫した en の値を反映できる．

ただし，Concatenation 後に同時に処理されるタプル群のうち en を変更するものは，動的コンパイル時に決定可能でなければならない．タプルに対する複数の処理が en を更新する場合には，複数単位演算ユニットブロックに分割することで，図 18 同様に Concatenation を実現できる．

4.　評　　価

本章では，提案する DR-SPE を評価する．評価のために DR-SPE を**表 4** に示す FPGA に実装した．開発ツールとして XilinxISE 12.1 Logic Edition を用い，合成には XST コンパイラを使用する．

評価対象として，**表 5** に示すパラメータの DR-SPE を想定する．このパラメータは，入力ストリームとして Streams on Wires [1] で想定されているアプリケーション同様に，シン

表 4　XC6VLX240T-1 の主な仕様
Table 4　A specification of XC6VLX240T.

項目	個数
#. of Slice Registers	301,440
#. of Slice LUTs	150,720
#. of Slices	37,680
#. of BRAM（32 KB）	416
#. of DSP48	768

表 5　合成する DR-SPE のパラメータ表
Table 5　Parameters for synthesized DR-SPE.

項目	値
Tuple bit width	96 bit
Operator bit width	32 bit
#. of units in a block	8
max way for Union	8

ボル，時刻および価格の 3 個の値からなるタプルを処理することを想定して決定している．シンボルは，“UBSN” のような 4 Byte（= 32 bit）の文字列，時刻および価格をそれぞれ 32 bit の数で表すと，タプルのビット幅（Tuple bit width）は 96 bit となる．プロセッサの演算ビット幅（Operator bit width）はこれらの値を自然に演算できる 32 bit とする．また，図 1 に示した Streams on Wires の Q4 では 5-way の Union が用いられているため，ストリーム入出力制御器の設定で 5-way の Union を実現するにあたり必要十分であるよう，ブロック内の単位演算ユニットの個数（#. of units in a block）を，これを満たす 2 の冪乗の数である 8 個とした．この場合，実現可能な Union の最大 way 数（max way for Union）は 8 である．

本章では，まず，［設計課題 3］に示した再構成時間の最小化を実現しているか，クエリの構成に要する時間を評価する．この時間は，ユーザが DR-SPE を制御するホストコンピュータから DR-SPE へ，所望のクエリを構成するために設定すべきデータを転送する時間である．設計したアーキテクチャのレジスタの個数から設定にかかる時間を算出し，また実験的に確かめる．次に，［設計課題 1］および［設計課題 2］をそれぞれ実現できているか，回路面積および信号遅延を Streams on Wires との比較によって評価する．

4.1　クエリの構成時間の評価

提案する DR-SPE では，DR-SPE 構成する演算構成要素である単位演算ユニット，スイッチボックスおよびストリーム入出力制御機構，それぞれが備える設定レジスタに適切な値を書き込むことで，所望のクエリを実現する．したがって，動的再構成に要する時間は，それらのレジスタに値を設定するために必要なデータのサイズおよび通信帯域によって決定される．

DR-SPE の動作を設定するために必要なデータサイズ $DR\text{-}SPE_{config}$ bit は，単位演算ユニットの動作の設定に必要なデータサイズ op_{config} bit，スイッチボックスの動作の設定に必要なデータサイズ $switchbox_{config}$ bit，ストリーム入力制御器の動作の設定に必要なデータサイズ eis_{config} bit，および，ストリーム出力制御器の動作の設定に必要なデータサイズ eos_{config} bit の和に等しい．すなわち，

$$DR\text{-}SPE_{config} = op_{config} + switchbox_{config} + eis_{config} + eos_{config} \quad (1)$$

である．以下，各演算構成要素の設定に必要なデータサイズの算出式を述べる．

単位演算ユニットの動作は，ALU，ALU への入力を選択する selector，出力を切り換える MUX の設定で決定される．ここで，$tuple_{width}$ および op_{width} をそれぞれ，取り扱うタプル bit 幅および ALU の演算 bit 幅とするとき，単位演算ユニット中で ALU に入力するデータを選択する selector を設定するために必要なデータサイズ（$op\_selector_{config}$ bit）は，入力候補を識別するために必要な bit 数に等しい．selector の入力候補は，入力タプルの部分ビット列 $tuple_{width}/op_{width}$，result，en および variable である．したがって，selector を設定するために必要なデータサイズは

$$op\_selector_{config} = \lceil \log_2(tuple_{width}/op_{width} + 1 + 1 + 1) \rceil \quad (2)$$

で算出される．これに出力データのソースを選択する MUX を制御するための信号（$tuple_{width}/op_{width} + 1 + 1 + 1$ bit），ALU の命令選択（4 bit），および内部変数の初期値（op_{width} bit）を加えると，単位演算ユニットに所望の動作を設定するために必要なデータサイズ（op_{config} bit）は，

$$op_{config} = op\_selector_{config} + tuple_{width}/op_{width} + 1 + 1 + 1 + 4 + op_{width} \quad (3)$$

で算出される．

DR-SPE のスイッチボックスはユーザが設定した期間に応じて入力ソースを 2 方向から自動的に選択する機構を有する．したがって，スイッチボックスを設定するために必要なデータサイズ $switchbox_{config}$ bit は，それぞれの入力ソースに対し，7 方向からの入力ソースの方向の選択に 3 bit と切り換えるまでの期間を設定するカウンタの最大値 $\lceil \log_2(way) \rceil$ bit のサイズのデータが必要である．ここで，way は DR-SPE が実行可能な Union の最大ウェイ数である．したがって，$switchbox_{config}$ bit は，

$$switchbox_{config} = 2 * (3 + \lceil \log_2(way) \rceil) \quad (4)$$

で算出される．

ストリーム入力器のデータサイズ eis_{config} bit は，入力を判定するためのデータ元を選択するための selector に必要なデータサイズ $s\_selector_{config}$ bit が，

$$s\_selector_{config} = \lceil \log_2(tuple_{width}/op_{width} + 1 + 1) \rceil \quad (5)$$

で算出される．これに出力データの入力元を選択する信号（2 bit），反転/非反転を選択する信号（1 bit），およびカウンタの初期値と最大値（それぞれ $\lceil \log_2(block) \rceil$ bit）を加えて，

$$eis_{config} = s\_selector_{config} + 2 + 1 + 2 * \lceil \log_2(block) \rceil \quad (6)$$

で算出される．ここで $block$ は，入出力ストリーム制御器を共有する単位演算ユニットグループ中の単位演算ユニットの個数を示す．

同様に，ストリーム出力器のデータサイズ eos_{config} bit は，セレクタに必要なデータサイズ $s\_selector_{config}$ bit に，出力データの入力元を選択する信号（2 bit），反転/非反転を選択する信号（1 bit），カウンタの初期値と最大値（それぞれ $\lceil \log_2(block) \rceil$ bit），および result 入力による制御の有効/無効を制御する信号（1 bit）を加えて，

$$eos_{config} = s\_selector_{config} + 2 + 1 + 2 * \lceil \log_2(block) \rceil + 1 \quad (7)$$

で算出される．

表 5 に示したパラメータの DR-SPE を構成する場合，単位演算ユニット，スイッチボックス，ストリーム入力制御器およびストリーム出力制御器を設定するためのデータサイズは，式 (1)〜(7) を用いて，それぞれ，48 bit，12 bit，12 bit および 13 bit の計 85 bit と得られる．構成情報の転送帯域を σbps とすると，構成に必要な時間 t 秒は，

$$t = 85/\sigma \quad (8)$$

で得られる．すなわち，構成に必要なデータを転送する時間は σ が 1 M であれば 85 μ 秒，10 M であれば 8.5 μ 秒と算出できる．各単位演算ユニット，スイッチボックス，ストリーム入出力制御器のレジスタへ 1 Mbps でデータを転送可能な再構成データパスを実装したところ，データ転送開始から設定完了までに 8,500 サイクルを要した．実験環境の動作周波数が 100 MHz であることから，8,500 サイクルは 85 μ 秒に相当する．以上より構成時間は理論的にも実験的にも等しく 85 μ 秒だった．したがって，提案プロセッサアーキテクチャは，新規クエリ登録や動的クエリ最適化をたかだか 85 μ 秒で実行できる．

4.2　回路面積と信号遅延のオーバヘッド

DR-SPE は，動的再構成機構を実現するために，単にストリーム処理エンジンを構成する場合に比べ，余分なハードウェアリソースを必要とする．本節では，DR-SPE を表 4 に示した XC6VLX240T-1 上に実装するときに必要なハードウェアリソース量と，直接ストリーム処理エンジンを構成する Streams on Wires が必要とするハードウェアリソース量を比較した結果を示す．

表 6　単位演算ユニットのハードウェアリソース使用量
Table 6　A result of resource usage for operation unit.

項目	使用量
#. of Slice Registers	181
#. of Slice LUTs	483

表 7　スイッチボックスのハードウェアリソース使用量
Table 7　A result of resource usage for switchbox.

項目	使用量
#. of Slice Registers	22
#. of Slice LUTs	285

表 8　ストリーム入出制御器のハードウェアリソース使用量
Table 8　A result of resource usage for stream input controller.

項目	使用量
#. of Slice Registers	24
#. of Slice LUTs	74

表 9　10 × 10 の単位演算ユニットをタイル状に敷き詰めた DR-SPE のハードウェアリソース使用量
Table 9　A result of resource usage for DR-SPE including 10 × 10 operation units.

項目	使用量	使用率
#. of Slice Registers	20,038	6 %
#. of Slice LUTs	88,421	56 %

設計した単位演算ユニット，スイッチボックス，およびストリーム入出力制御器の合成して得られた回路のハードウェアリソース使用量を**表 6**，**表 7**，および**表 8** に示す．最高動作周波数は，それぞれ，232.8 MHz，430.3 MHz，および 440.335 MHz となった．ただし，ストリーム入出力制御器のハードウェアリソース量に CAM の容量は含んでいない．また，**表 9** に，10 × 10 の単位演算ユニットをタイル状に敷き詰めた DR-SPE を構成する場合のハードウェアリソース使用量を示す．このとき，最高動作周波数は 172.1 MHz となった．

一方で，図 1 に示すクエリ Q1〜Q4 を，Streams on Wires に示されているように，Algebra をビルディングブロックとして静的に FPGA 上に構成する場合の回路のハードウェ

表 10　Algebra を静的に合成する Streams on Wires のハードウェアリソース使用量
Table 10　A result of resource usage for static queries by using Algebra.

Query	#. of Slice Registers	#.of LUTs
Q1	202	8
Q2	270	10
Q3	585	272
Q4	698	283

表 11　必要な単位演算ユニット数
Table 11　The number of required operation units for each query.

Query	#. of units	#. of Slice Registers	#.of LUTs
Q1	2	362	966
Q2	4	724	1,032
Q3	11	1,991	5,313
Q4	15	2,715	7,254

アリソース使用量を**表 10** に示す．また，Q1〜Q4 を DR-SPE で合成する場合に必要となる単位演算ユニットの個数および対応するハードウェアリソース量を**表 11** に示す．なお，Q5 は本論文の比較対象外であるため，実装を行っていない．比較対象外である理由は次のとおりである．本節の目的は，DR-SPE の再構成機構を実現するために必要なハードウェアリソース量および信号遅延を評価することである．Q5 のクエリを実現するには CAM（連想メモリ）が必要であり，CAM の実装に必要なハードウェアリソース量および信号遅延は，そのアドレス幅およびデータ幅に大きく影響される．したがって，本論文で評価するべき，再構成機構実現に必要なハードウェアリソース量および信号遅延の増分の程度が CAM の設定によって変わってしまう．CAM のアドレス幅，データ幅を大きくすると再構成機構実現に必要なハードウェアリソース量および信号遅延の増分は小さく，逆に，CAM のアドレス幅，データ幅を小さくすると再構成機構実現に必要なハードウェアリソース量および信号遅延の増分は大きくみえてしまい，正しく評価することができない．したがって，本論文では Q5 を比較対象外とした．

表 10 および表 11 より，DR-SPE では，Streams on Wires に比べ，レジスタ使用量は約 1.8 倍〜約 3.9 倍，LUT 使用量は約 19.5 倍〜約 120.8 倍であることが分かる．LUT は，DR-SPE における動的再構成を実現するためのマルチプレクサを構成する必要があり，Streams on Wires に比べ大量に必要となることが分かる．一方で，レジスタ増加分は，LUT の増加分に比べてはるかに少ない．その観点からすれば，Streams on Wires と DR-SPE では，レジスタの使用量はほぼ同数と見なせる．なお，DR-SPE のレジスタ使用量が多い理由は，中間結果の使用しないビット用レジスタと動作設定用レジスタの増加による．タプルや中間結果を保持するレジスタは，Streams on Wires と DR-SPE のいずれにおいても必要であるため，増分はきわめて少なくなっている．

静的に構成する Streams on Wires でのクエリと DR-SPE によるクエリの処理性能を比較する．Q1〜Q4 を同じく XC6VLX240T を用いて，静的に構成すると，それらの最高動作周波数は総じて 200 MHz 程度であった．DR-SPE の最高動作周波数は 172.1 MHz であるから，最高動作周波数は低下する．ここで，Streams on Wires および DR-SPE のスループットは，どちらも最大 1 tuple/サイクルである．したがって，1 tuple が 96 bit の場合，Streams on Wires と DR-SPE の最大スループットは，それぞれ，19,200 Mbps と 16,521 Mbps となる．ただし，このスループットは実験室で作成できるデータ到着率（1,000,400 packets/s）[1] の限界を上回る．したがって，提案する DR-SPE と Streams on Wires が提供しうるデータソースとアプリケーションのシステム間での性能は等しいと考えられる．

ここで，最高動作周波数は，構成した回路の同期回路中の組合せ回路に含まれる素子の数によって決定される．DR-SPE では，どんなクエリを実現しようとも，ハードウェア回路そのものが変更されるわけではないため，動作周波数は変化しないことに注意されたい．したがって，最高動作周波数の範囲内でユーザが設定した周波数相当のスループットが保証される．

スループットが同等である一方で，DR-SPE のレイテンシは，Q1，Q3，Q4 を静的な処理エンジンとして構成する場合に比べて 1〜3 サイクル増加した．この理由は，Union を行うために単位演算ユニットを多段に組み合わせる必要があること，および単純な組合せ回路を 1 サイクル内で実行できるように組み合わせるという最適化ができないことによる．しかし，100 MHz で処理エンジンが動作する場合，1〜3 サイクルのレイテンシの増加は 10 n 秒〜30 n 秒の増加に相当する．これは通信インタフェースなどの，その他の部分のレイテンシに比べて小さいため，アプリケーションにほぼ影響がないと考えられる．

ただし，DR-SPE では，ASIC 化により，これらの欠点を改善可能である．一方，Streams on Wires は ASIC 化が現実的でない．したがって ASIC 化を視野に入れれば，DR-SPE は Streams on Wires より高い性能を実現できる．この詳細は，4.3 節で議論する．

動的再構成可能な機構を有する DR-SPE の利点を，所望のクエリの候補をすべて FPGA 上にあらかじめ実装しておく手法（Streams on Wires）と比較しつつ考察する．Streams

on Wires と DR-SPE の処理性能を比較するために実装した Q1〜Q4 のクエリ（図 1）に必要なハードウェアリソース量は表 10 に示したとおりである．この値は，Q1〜Q4 を同時に FPGA に実装可能であることを意味する．したがって，Q1〜Q4 だけをユーザが利用するならば，これらのクエリはあらかじめ FPGA 上にモジュールとして用意されることで実現される．その場合，スイッチボックスやストリーム入出力制御器は不要であり，データパス中の余分なビットも不要になる．しかし，この方式は 2 章で述べたクエリの動的最適化に対応できない．なぜなら，クエリを構成する演算子の個数が k 個であれば，k 個の演算子を並べ換えた結果である k!通りの処理木すべてをあらかじめ FPGA 上に実装する必要があるからである．これは明らかに非現実的である．DR-SPE では，評価に用いた FPGA 上に表 9 に示したように 10 × 10（= 100）の演算器を並べられる．したがって，DR-SPE では，100 個の選択演算からなるクエリの選択演算を並べ換えて得られる 100!通りの処理木を実現可能である．

4.3　ASIC としての実装についての議論

提案する DR-SPE は，FPGA 上への実装だけではなく，カスタム IC である ASIC として実装することも現実的に可能である．ASIC に DR-SPE を実装する場合，FPGA 上に所望の回路を構成する場合に比べ，回路面積と消費電力の削減および動作周波数の向上が実現できる．一般に，ASIC の作成には高い開発コストおよびその資金が必要であるため，ASIC 化が現実的である場合は，大量に 1 種類のハードウェア回路が作成できる場合に限定される．

ストリーム処理エンジンでは，ユーザによって実行したいクエリが異なる．したがって，ASIC として実装される専用ストリーム処理エンジンでは，ユーザごとにクエリをカスタマイズできなければならない．DR-SPE では，図 4 に示すように，入出力タプルの幅や ALU の演算幅を，対象とするストリームデータすなわちアプリケーションに応じてカスタマイズした後は，ハードウェア回路を変更する必要がない．所望のクエリは，内部のレジスタの値を外部から適切に設定できる機構を用いて定義される．レジスタの値の設定はハードウェア回路そのものを変更するわけではないため，この機構は ASIC 化しても損われることはない．すなわち，たとえば，IP アドレスに対するアプリケーションなど，同じタプルの bit 数のデータを扱うアプリケーションに対してコンフィギュレーションした DR-SPE を作成することができる．

一般に，ユーザごとに実行したいクエリの種類を Q_{num}，データの種類を D_{num} とすると，それぞれともに正の整数であるから，

$$Q_{num} \times D_{num} > D_{num} \tag{9}$$

が成り立つ．したがって，所望のクエリに合わせたストリーム処理エンジンを専用ハードウェアとして作成する場合に開発しなければならないマスクの種類（$Q_{num} \times D_{num}$）に比べ，DR-SPE を用いて所望のクエリを実装する場合に開発しなければならないマスクの種類（D_{num}）は少ない．マスクの開発には大きなコストが必要となるため，開発しなければならないマスクの種類の削減は開発コスト削減につながる．また，専用ハードウェアとして設計された 1 つのストリーム処理エンジンでサポート可能なユーザに比べ，1 つの DR-SPE でサポート可能なユーザの数は多い．そのため，DR-SPE は 1 つのデバイスを多数のユーザ向けに大量生産することに適しており，低価格化が可能となる．ただし，以上の議論は，適用分野における Q_{num} について D_{num} 定量的に示せているわけではない．

一方で，Streams on Wires では，FPGA というコンフィギュラブルなハードウェアを前提として，アプリケーションおよびユーザの所望するクエリに対するストリーム処理エンジンを構成できる．構成されるストリーム処理エンジンは多様性を持ち，それぞれを ASIC として作成するための開発コストと資金に見合うだけの汎用性が得られない．したがって，Streams on Wires を ASIC として作成することは，現実的でない．

5.　まとめと今後の課題

本論文では，動的再構成可能機構を有するストリーム処理エンジンの設計課題として面積増加の抑制，信号遅延増加の抑制および再構成時間の最小化の 3 つを示し，これらを解決する動的再構成可能ストリーム処理エンジン DR-SPE のプロセッサアーキテクチャを提案した．DR-SPE は，単位演算ユニット，スイッチボックスおよびストリーム入出力制御器を用いることで Streams on Wires と同等の演算子を実現できる．

XC6VLX240T-1 を用いて，提案プロセッサアーキテクチャを合成，配置配線して，DR-SPE における再構成時間，回路面積および信号遅延を評価した．その結果，再構成時間は構成に必要なデータサイズから算出した理論値どおり，DR-SPE では演算子実行順序の切換え処理を 85 μ 秒で実現できることが分かった．これはアルゴリズムトレードで利用するには十分な性能である．また，回路面積に関しては，提案するアーキテクチャにおけるレジスタの個数は Streams on Wires とほぼ同等である一方で，LUT が大量に必要となることが分かった．信号遅延は Streams on Wires よりやや増加するものの，システムとしては同程度のスループットを実現できることが分かった．さらに，LUT の増加および信号遅延の増加については ASIC 化によって改善できることを議論した．

今後の課題はコンパイル時間の短縮と生成されるコードの質を高めるために，コンパイル

アルゴリズムの最適化技法を開発することである．

謝辞　本研究の一部は科研費（#22700090），新世代ネットワーク技術戦略の実現に向けた萌芽的研究，を受けたものである．ここに記して謝意を表す．

参 考 文 献

1) Mueller, R., Teubner, J. and Alonso, G.: Streams on wires: A query compiler for FPGAs, *Proc. VLDB Endow.*, Vol.2, No.1, pp.229–240 (2009).
2) Mueller, R., Teubner, J. and Alonso, G.: Glacier: A query-to-hardware compiler, *SIGMOD '10: Proc. 2010 International Conference on Management of Data*, pp.1159–1162, New York, NY, USA, ACM (2010).
3) arrowhead とは？http://www.tse.or.jp/rules/stock/arrowhead/info.html
4) Cisco CarrierRouting System, http://www.cisco.com/en/US/products/ps5763/index.html
5) Gold, B., Ailamaki, A., Huston, L. and Falsafi, B.: Accelerating database operators using a network processor, *Proc. 1st International Workshop on Data Management on New Hardware*, DaMoN '05, New York, NY, USA, ACM (2005).
6) Govindaraju, N., Gray, J., Kumar, R. and Manocha, D.: GPUTeraSort: High performance graphics co-processor sorting for large database management, *Proc. 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pp.325–336, New York, NY, USA, ACM (2006).
7) Gedik, B., Yu, P.S. and Bordawekar, R.R.: Executing stream joins on the cell processor, *Proc. 33rd International Conference on Very Large Data Bases*, VLDB '07, pp.363–374, VLDB Endowment (2007).
8) Mueller, R., Teubner, J. and Alonso, G.: Data processing on FPGAs, *Proc. VLDB Endow.*, Vol.2, pp.910–921 (Aug. 2009).
9) Woods, L., Teubner, J. and Alonso, G.: Complex Event Detection at Wire Speed with FPGAs, *PVLDB*, Vol.3, No.1, pp.660–669 (2010).
10) DeWitt, D.J.: DIRECT – a multiprocessor organization for supporting relational data base management systems, *Proc. 5th Annual Symposium on Computer Architecture*, ISCA '78, pp.182–189, New York, NY, USA, ACM (1978).
11) 関野　陽，植村俊亮：データベース・マシン．情報処理，Vol.17, No.10, pp.940–946 (1976).
12) Avnur, R. and Hellerstein, J.M.: Eddies: Continuously adaptive query processing, *SIGMOD Rec.*, Vol.29, pp.261–272 (May 2000).
13) Eurviriyanukul, K., Paton, N.W., Fernandes, A.A.A. and Lynden, S.J.: Adaptive join processing in pipelined plans, *Proc. 13th International Conference on Extending Database Technology*, EDBT '10, pp.183–194, New York, NY, USA, ACM (2010).
14) Mei, B., Vernalde, S., Verkest, D. and Lauwereins, R.: Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study, *Proc. Conference on Design, Automation and Test in Europe – Volume 2*, DATE '04, pp.21224–21229, Washington, DC, USA, IEEE Computer Society (2004).
15) 津野田賢伸，高田雅士，秋田庸平，田中博志，佐藤真琴，伊藤雅樹：ディジタルメディア向け再構成型プロセッサ FE-GA の概要（アーキテクチャII，デザインガイア-VLSI設計の新しい大地を考える研究会-）．電子情報通信学会技術研究報告．RECONF，リコンフィギャラブルシステム，Vol.105, No.451, pp.37–41 (2005).

(平成 22 年 12 月 20 日受付)
(平成 23 年 4 月 7 日採録)

(担当編集委員　吉田 尚史)

三好　健文（正会員）

2003 年東京工業大学工学部電気電子工学科卒業，2005 年同大学院電子機能システム専攻修士課程修了．2007 年同大学院物理情報システム専攻博士課程修了．博士（工学）．同年東京大学情報理工学系研究科特任助教．東京工業大学情報理工学研究科産学官連携研究員を経て，2009 年より電気通信大学大学院情報システム学研究科助教．コンパイラ，HW/SW 協調設計に関する研究に従事．IEEE，ACM，電子情報通信学会各会員．

寺田　祐太

2011 年電気通信大学電気通信学部卒業．同年株式会社アバールデータへ入社．組み込みシステム開発に従事．

川島　英之（正会員）

1999 年慶應義塾大学理工学部電気工学科卒業．2005 年同大学院理工学研究科開放環境科学専攻後期博士課程修了．同年慶應義塾大学理工学部助手．2007 年筑波大学大学院システム情報工学研究科講師，ならびに計算科学研究センター講師．博士（工学）．センタデータ管理に関する研究に従事．日本データベース学会，ACM，IEEE 各会員．

吉永　　努（正会員）

1986 年宇都宮大学工学部情報工学科卒業．1988 年同大学院工学研究科修士課程修了．同年より宇都宮大学工学部助手．1997 年から翌年にかけて電子技術総合研究所客員研究員．2000 年電気通信大学大学院情報システム学研究科助教授．現在，同教授．博士（工学）．並列分散処理，計算機アーキテクチャ等に興味を持つ．電子情報通信学会，IEEE 各会員．