動的再構成可能データベース処理エンジンとクエリコンパイラの検討

三好健文[†1] 寺田祐太[†2]
川島英之[†3] 吉永努[†1]

本稿では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．ストリーミングデータベースでは，時々刻々と変化するデータの流れであるストリーミングに，SQL で記述した命令で検索や算術演算などを適用できる．データを取り零すことなくクエリを実行するための計算機への演算負荷は大きい．

提案する DR-SPE は，専用ハードウェアによるストリーム処理エンジンであり，並列処理による高い処理性能と，所望のクエリの実行や動的問い合わせ最適化を実現できる柔軟性の両方を実現する．DR-SPE は，ストリーミングデータベースの処理を行うための基本的な演算構成要素を多数持ち，それらを組み合わせることで所望のクエリを実行する．実現可能な処理機能は，Streams on Wires[1)] で提案されている FPGA 上に静的に構成されるストリーム処理エンジンと同等である．

本稿では，DR-SPE のアーキテクチャ設計および FPGA XC6VLX240T-1 を用いて実装する場合の回路規模および信号遅延を示す．合成の結果は，10×10 程度の要素が FPGA 上に配置でき，その最高動作周波数が 172.1MHz であることを示す．また，演算構成要素を変更するために必要な情報は最小で 85bit であり，高速に更新できる．

A Dynamic Reconfigurable Database Processing Engine and Its Compiler

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

A dynamic reconfigurable streaming processor engine DR-SPE is proposed. Computation power of a streaming database utilizing SQL should be large enough to avoid processing losses.

The proposed DR-SPE achieves high computational performance by employing parallelism and high flexibility for execution of various query plans and optimizing them at run time. DR-SPE consists of a lot of processing elements for execution of streaming database, and query plans are executed by their combinations. Functions supported by DR-SPE equal to the ones supported by Streams on Wires[1)], which is a configurable streaming processor engine performed on FPGA statically.

In this paper, the architecture of DR-SPE is described. The implementation of DR-SPE on a FPGA (XC6VLX240T-1) shows that we can place 10×10 operation units on it and the maximum frequency is 172.1MHz. The minimum configuration data are 85bit, so that DR-SPE is reconfigurable in short time.

1. はじめに

時々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挙げられる．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，取り零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，これを処理する為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

†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

表 1 各クエリの合成と配置配線にかかる時間

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

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．2.80GHz で動作する Core^{TM} i7 860，メモリ搭載量 8GiB の Linux(Fedora 13) 計算機上で，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．従って 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．

そこで本稿では，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する方式を提案する．この方式では，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．この方式を採用した SPE を，本論文では動的再構成可能ストリーム処理エンジン (DR-SPE) と表記する．

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．さらに，再構成の情報を動的に生成するために，短時間で最適化されたコードを生成可能なコンパイラ技術が求められる．

本稿では DR-SPE の実現を目指し，プロセッサアーキテクチャ設計およびクエリコンパイラの初期検討を行う．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．

本稿の構成は次の通りである．第 2 節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．第 3 節では，DR-SPE の設計課題と提案アーキテクチャを述べ，提案アーキテクチャが Streams on Wires と同等の処理機能を有することを示す．第 4 節では提案アーキテクチャ上にクエリを実現するために必要なクエリコンパイラについて述べる．第 5 節では，提案アーキテクチャの評価として FPGA 上に実装した場合のハードウェアリソース量および信号遅延，ならびに再構成時間の見積りを示す．最後に第 6 節では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，監視カメラ等，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 2ms 程度で株価を生成し，ルータは 300Tbps 程度でデータを生成し，一般的な産業用ロボットは 1ms 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．例えば，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辿っている．

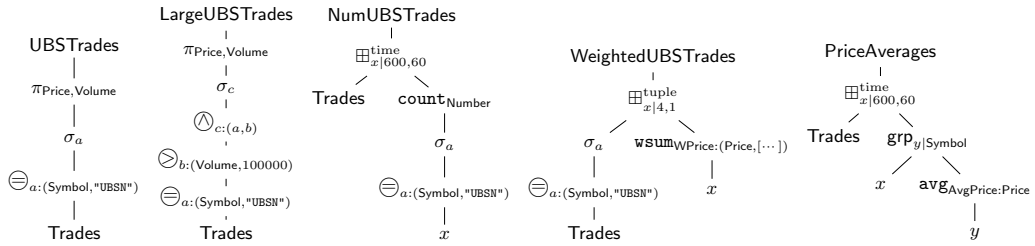
ストリームデータ処理において最も重要な研究課題の一つに，大規模ストリームデータに対するスケーラブルな処理がある．性能向上のために，FPGA を用いてストリーム処理を高性能化する既存研究[1)3)4)] がある．

一方，アルゴリズム的な観点からスケーラブルな処理を実現するために極めて重要な役割を果たす技術として，動的クエリ最適化がある．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，処理性能を向上する技術である．関係データベースにおけるクエリ最適化は動的である必要がない一方で，ストリームデータ処理においては，刻一刻とシステムに到着するタプルを処理する為に動的であることが求められる．タプル到着に伴って，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．

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

⋆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．



(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[1] に例として挙げられているクエリ (文献 1) より引用)

表 2 Streams on Wires[1] において定義される演算要素である Algebra(文献 1) より引用)

$\pi_{a_1,\ldots,a_n}(q)$	projection
$\sigma_a(q)$	select tuples where field a contains true
$\circledast_{a:(b_1,b_2)}(q)$	arithmetic/Boolean operation $a = b_1 \star b_2$
$q_1 \cup q_2$	union
$agg_{b:a}(q)$	aggregate agg using input field a, $agg \in \{\texttt{avg}, \texttt{count}, \texttt{max}, \texttt{min}, \texttt{sum}\}$
$q_1\ \texttt{grp}_{x\|c}\ q_2(x)$	group output of q_1 by field c, then invoke q_2 with x substituted by the group
$q_1 \boxplus^t_{x\|k,l} q_2(x)$	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
$q_1 \circledcirc q_2$	concatenation; position-based field join

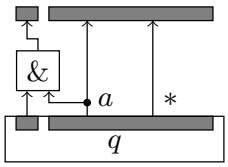


図 2 ビルディングブロックとしての Algebra 回路 (文献 1) より引用)

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ーフェイ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．

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．

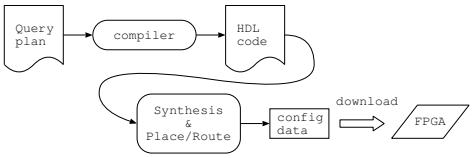


図 3 Streams on Wires[1] 処理におけるエンジン生成フロー

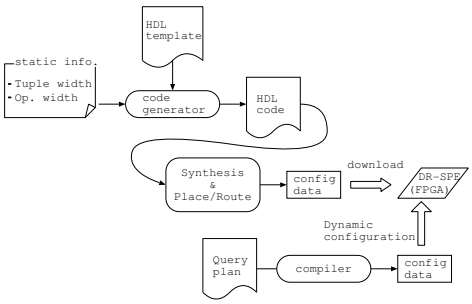


図 4 動的再構成可能ストリーム処理エンジン (DR-SPE)

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．従って，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い合わせ最適化の処理の実現は困難である．

そこで本稿では，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，FPGA 上に構成し

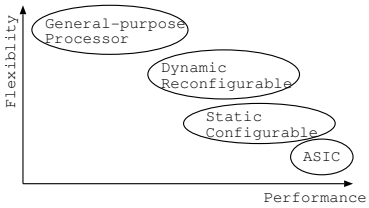


図 5 動的再構成可能ストリーム処理エンジン (DR-SPE) の位置付け

た DR-SPE 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，得られた構成情報が FPGA に書き込まれる．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．

DR-SPE は次の特徴をもつ．

- 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，ならびにパラメタを設定が可能

この特徴の為，**図 5** に示すように，汎用プロセッサ上のソフトウェアに比べてサイクルレベルでの処理とデータ/パイプライン並列性による高い処理性能を達成すると共に，動的なクエリの追加や変更という柔軟性を達成する．

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．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．その為，構成情報を生成するための計算コストが大幅に削減される．また，クエリプラン変更時には，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ため，データ転送時間と更新時間を Streams on Wires に比べて大きく削減できる．

3.1 DR-SPE の設計課題

動的再構成可能なプロセッサの設計課題は，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 点がある．これらについて下記に述べる．

第一に，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，== 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するために，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．

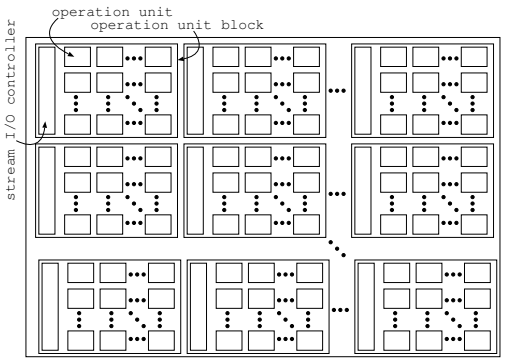


図 6 DR-SPE アーキテクチャ概要

第二に，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言える．クエリプランから 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．従って，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を鑑みれば，実現不可能であると考えられる．

第三に，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 提案アーキテクチャ

図 6 に，提案する DR-SPE の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

算ユニットで構成される．単位演算ユニット同士はスイッチボックス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．ストリーム入出力制御器は，各ブロックに一つずつある．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．スイッチボックスでは，単位演算ユニット間の接続を静的に決定するだけではなく，サイクルによって複数の入力を動的に切り換えられる．これにより，その時々に応じてパスを変更する適応的な処理も実現可能である．

DR-SPE のアーキテクチャは前述の設計課題を解決するために次の特徴を持つ．

- 各構成要素では，単一サイクルの処理のみをサポート
- データバス幅は固定で，コンパイラによってスケジューリング

提案アーキテクチャを構成する単位演算ユニットと制御機構では，単一サイクルで処理できる演算のみをサポートする．複雑な演算処理が必要な場合には，複数の構成要素を組み合わせることで実現する．従って 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は高くなる．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，最大スループットとして 1tuple/サイクルを保証できる．また，細粒度の構成要素では回路中の信号遅延を小さく抑えられ，最高動作周波数を高く保てる．

また，ハードウェアリソース使用量を抑えるために，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，固定幅としている．複数の値をユニット間で共有する必要がある場合，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備えなければならないという制約を回避している．

上記に加えて，単位演算ユニットの処理内容を単一サイクルで実行可能な処理に制限することと，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量が小さくまとめられる．従って，処理の実現に必要なパラメタが少なくなり，再構成にかかる時間を短縮できる．

所望の処理を細粒度の演算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，FE-GA[5] や ADRES[6] などがある．これらの多くは信号処理などを対象にしている為，処理におけるデータのパスは単純に構成できる．

一方で，ストリームデータ処理では，Union，Windowing，そして Grouping といった，複数の処理を統括しなければならない処理や，Aggregation のように単純な処理として展開することが困難な演算を数多く含む．従って，ストリームデータ処理を効率良く実行するには，ストリームを制御する機構が必要になる．このため，DR-SPE は，ストリームデータ処理エンジンを構成するために，十分に細かい粒度の単位演算ユニットとストリーム制御機構を兼ね備える．

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．

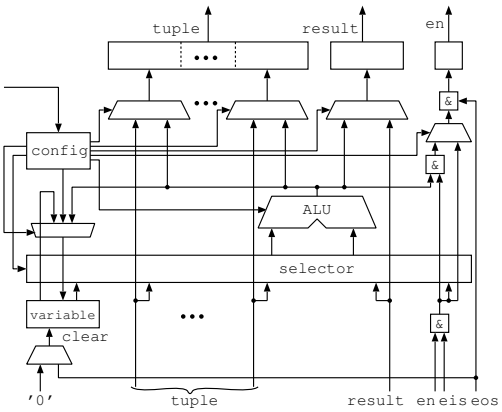


図 7 単位演算ユニット

表 3 単位演算ユニットで実現可能な演算

種別	演算
算術演算	和，差，+1，-1
シフト	左 1bit シフト (ローテート有/無)， 右 1bit 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	==，>，>=，!=

3.2.1 単位演算ユニット

図 7 に単位演算ユニット (Operation Unit) の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 は固定長のビット列として，この単位演算ユニットが FPGA 上に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 を一つ持つ．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充分小さい．従って，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．

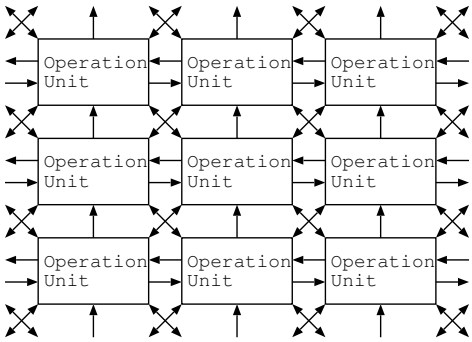


図 8 スイッチボックスによる単位演算ユニットの接続

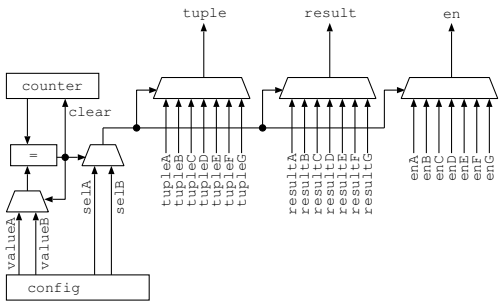


図 9 スイッチボックスのアーキテクチャ

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]の時，強制的に入力/出力信号の en が'0' に設定され，データは無効にされる．

3.2.2 スイッチボックス

スイッチボックスは，第 3.2.1 節で述べた単位演算ユニット同士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 8**)． どの方向からの入力を受けとるかは，設定によって，静的に固定するか，あるいは，指定した 2 方向からの入力を切り換えて与えるか，指定する．入力を切り換えて与える場合，そのタイミングはカウンタによって指定されたタイミングで自動的に切り換えられる．

アーキテクチャの詳細を**図 9** に示す．valueA，valueB，selA，selB の値は実行時に設定される．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同士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．従って，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 10** および**図 11** にストリーム入出力制御器を示す．これらは，一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，CAM データ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，常に入力データ中の 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．

⋆1 '0' および'1' は 1bit の信号がそれぞれ 0 あるいは 1 であることを表わす．

⋆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 3bit で定義できるため．

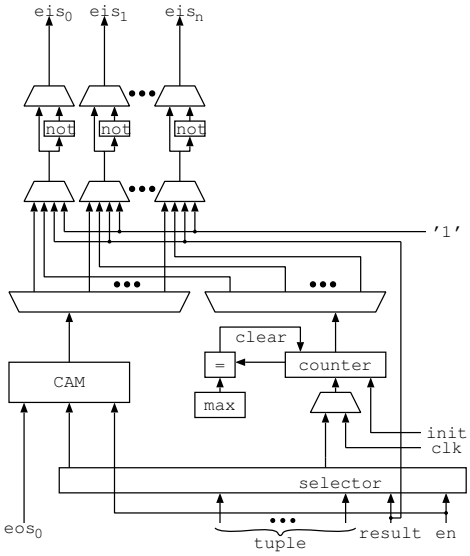


図 10 ストリーム入力制御器のアーキテクチャ

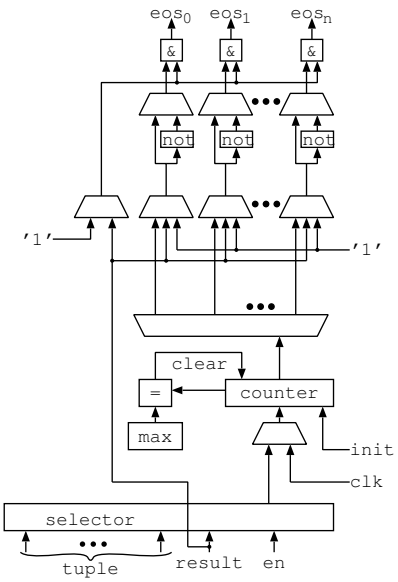


図 11 ストリーム出力制御器のアーキテクチャ

3.3 Algebra 実装例

DR-SPE の単位演算ユニットによって，Streams on Wires の Algebra 相当の処理を実現できることを，設定するパラメタと構成例によって示す．

3.3.1 Selection と Projection

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 12** に，result の 0bit 目を対象に Selection を適用する場合の単位演算ユニットの構成例を示す．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．

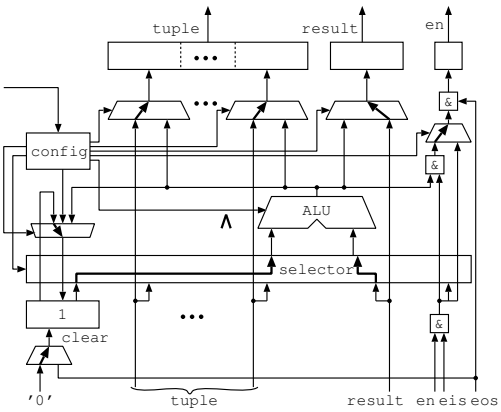


図 12 Selection の構成例

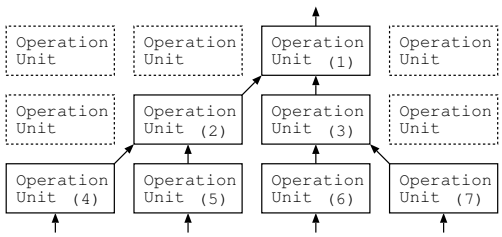


図 13 4-way Union の構成例

3.3.2 算術/論理演算

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 varilable に書き出される．

3.3.3 Union

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 13 に 4-way の Union の構成例を示す． (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 サイクルに一度入力元を切り換える．また，(2) の入力 (4) と (5) および，

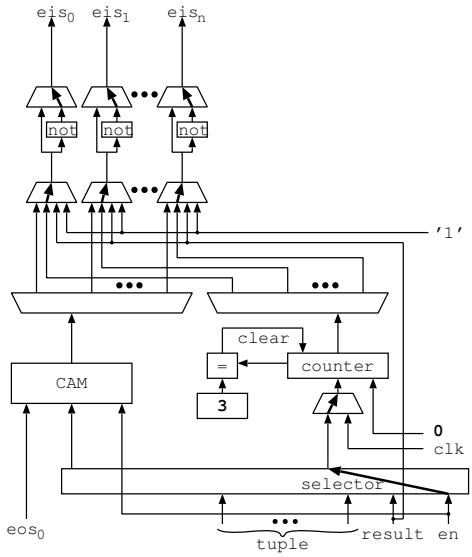


図 14 Windowing の実現例 (ストリーム入力制御器)

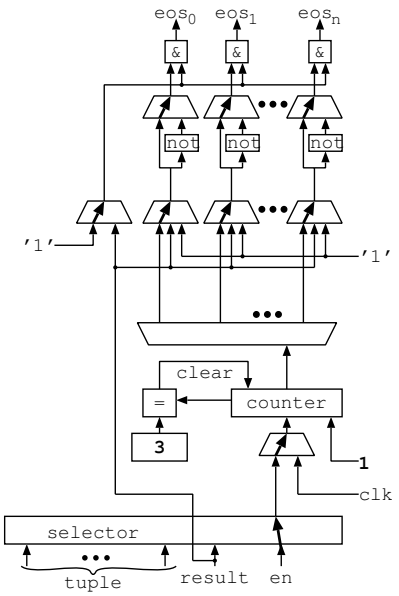


図 15 Windowing の実現例 (ストリーム出力制御器)

(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

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 14** および**図 15** にサンプルとなる構成例を示す．

図 14 および図 15 では，有効なデータが到達する度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

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．

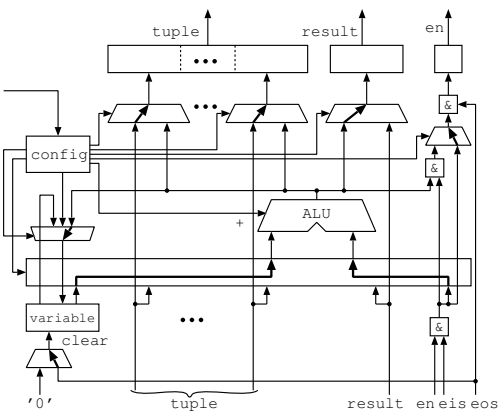


図 16 Aggregation の構成例

図 16 に Aggregation の例として有効データ (en が'1' のタプル) のカウンタの構成例を示す．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．この時，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

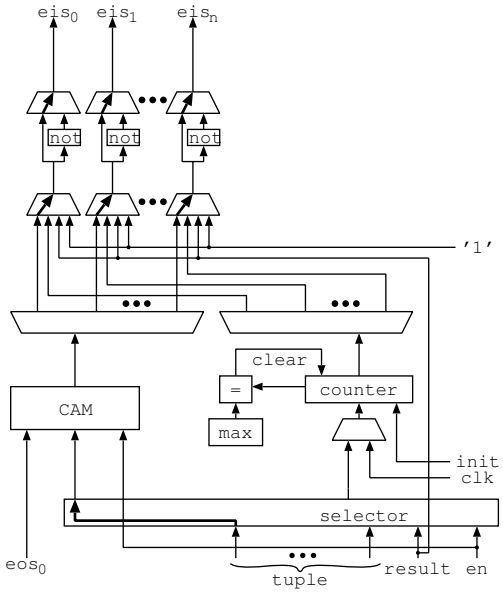


図 17 Grouping の構成例 (ストリーム入力制御器部分)

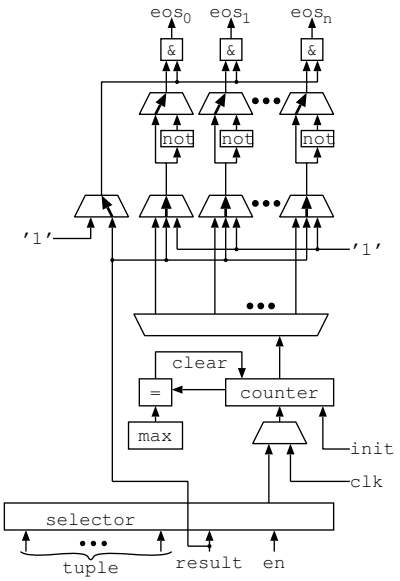


図 18 Grouping の構成例 (ストリーム出力制御器部分)

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み合わせが必要となる．

3.3.6 Grouping

Grouping はストリーム入出力制御器を活用することで実現される．**図 17** に Grouping を実現するためのストリーム入力制御器の構成例を示す．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構成例を**図 18** 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．

3.3.7 Concatenation

Concatenation を実現するためには，複数の単位演算ユニットを 1 まとまりとしてコンパイラで扱えばよい．ここで，Concatnation 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，1 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 19** に 2 つの入力タプルを Concatenation する処理の構成例を示す．図 19 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．

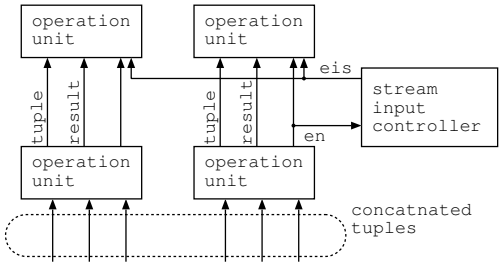


図 19 Concatnation の実現例

ただし，Concatnation 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 19 同様に Concatenation を実現できる．

4. クエリコンパイラ

DR-SPE では，所望のクエリを実現するための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器を提供する．実際にクエリを処理させるためにはクエリプランから，各

表 4 XC6VLX240T

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

演算構成要素内のマルチプレクサや定数に設定すべき値の構成情報を生成する必要がある．クエリプランから構成情報を生成するのが DR-SPE のクエリコンパイラである．

クエリコンパイラは，まず，与えられたクエリを解析し，クエリを構成する Algebra 集合に分割する．次に，クエリコンパイラは，Algebra とデータ授受関係をそれぞれノードおよびエッジとし，最終出力ノードをルートとするデータフローグラフを生成する．生成したデータフローグラフに基づき，DR-SPE 中の使用構成要素数および各種制約を考慮しながら，各 Algebra を単位演算ユニットへ配置し，接続関係を決定する．最後に，各要素のパラメタおよび接続関係情報である動的構成用の構成情報を出力する．

Union，Windowing，そして Grouping を行う処理では，対象となる複数のユニットの入出力制御を連携させる必要がある．このために，ストリーム入出力制御器を利用しなければならない．DR-SPE では，ストリーム入出処理器は，単位演算ユニットブロックあたり一つずつしか利用できないため，これらの処理はそれぞれ異なるブロックに配置するために分割する必要がある．また，Grouping は CAM を持つストリーム入出処理器を含むブロック上にしか構築できない．

一方で，算術演算や Selection などのストリーム入出処理器に対する制御が必要ない演算は，単位演算ユニットブロック中の単位演算ユニットの個数が許す限り，同一のブロックに配置できる．ハードウェアリソース使用率を考えると，できるだけ多くの単位演算ユニットを同一ブロックに配置されることが望ましい．

5. 評　　価

本節では，提案した DR-SPE を FPGA 上に実装し，必要となるハードウェアリソース量および信号遅延などについて評価する．まず評価環境について述べ，次に評価結果を示す．

5.1 評 価 環 境

DR-SPE を実装，評価する対象として Xilinx 社の FPGA である XC6VLX240T-1 を用いた．**表 4** に，XC6VLX240T の主な仕様を示す．開発ツールとして Xilinx ISE 12.1 Logic Edition を用い，合成には XST コンパイラを使用した．

合成する DR-SPE のパラメタを**表 5** に示す．

5.2 評 価 結 果

作成した単位演算ユニット，スイッチボックス，そしてストリーム入出力制御器の合成結果を**表 6**，**表 7**，

表 5 合成する DR-SPE のパラメタ表

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

表 6 単位演算ユニットのハードウェアリソース使用量

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

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

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

そして**表 8** に示す．最高動作周波数は，それぞれ，232.8MHz，430.3MHz，そして 440.335MHz 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．

10×10 の単位演算ユニットをタイル状に敷き詰めた DR-SPE を構成する場合の合成結果を**表 9** に示す．最高動作周波数は 172.1MHz となった．

一方で図 1 に示すクエリ Q1〜Q4 を，Algebra をビルディングブロックとして静的に FPGA 上に構成した場合の回路合成結果を**表 10** に示す．また，Q1〜Q4 を DR-SPE で合成する場合に必要だった単位演算ユニットの個数を**表 11** に示す．　タプルや中間結果を保持するレジスタは静的な回路でも動的再構成可能な回路でもほぼ同数であることがわかる．一方で，動的再構成を実現するためのマルチプレクサを実現する

表 9 ストリーム入出制御器のハードウェアリソース使用量

項目	使用量	使用率
#. of Slice Registers	20038	6 %
#. of Slice LUTs	88421	56 %

表 10 Algebra を静的に合成した場合のハードウェアリソース使用量

Algebra	項目	使用量
Q1	#. of Slice Registers	202
	#. of Slice LUTs	8
Q2	#. of Slice Registers	270
	#. of Slice LUTs	10
Q3	#. of Slice Registers	585
	#. of Slice LUTs	272
Q4	#. of Slice Registers	698
	#. of Slice LUTs	283

表 11 必要な単位演算ユニット数

Algebra	使用数
Q1	2
Q2	4
Q3	11
Q4	15

ために，多数の LUT が消費されることがわかる．

静的に構成したクエリと DR-SPE によるクエリの処理性能を比較する．Q1〜Q4 を静的に構成する場合，最高動作周波数は総じて 200MHz 程度であった．DR-SPE の最高動作周波数は 172.1MHz であるから，最高動作周波数は低下する．しかし，どちらも 100MHz での動作は可能であり，また，最大 1 tuple/サイクルで処理できることから，Streams on Wires 同様，DR-SPE は 100M tuple/秒での処理スループットが実現できることがわかる．また，DR-SPE では，どんなに複雑なクエリを実行する場合でもハードウェアの最高動作周波数に変化はないという優れた性質を有する．

処理スループットが同等である一方で，DR-SPE の処理レイテンシは，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 1〜3 サイクル増加した．この理由は，Union を行うためにユニットを多段に組み合わせる必要があること，および単純な組み合わせ回路を 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，100MHz 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 10n 秒〜30n 秒の増加に相当する．これは通信インターフェイスなどの，その他の部分の遅延に比べて小さい為，アプリケーションにほぼ影響がないと考えられる．

また，動的再構成に要する時間について評価する．DR-SPE 上の単位演算ユニット，スイッチボックスおよびストリーム入出力制御機構の構成情報は，それぞれ 48bit，14bit および 23bit の計 85bit である[⋆1]．構成情報の転送速度を σbps とすると，構成に必要な時間 t 秒は，およそ

$$t = 85/\sigma \quad (1)$$

で得られる．たとえば，σ が 1M であれば 85 μ秒，10M であれば 8.5 μ秒である．従って提案アーキテクチャは，新規クエリ登録や動的クエリ最適化を実現するのに十分な性能を，動的再構成という観点において有すると言える．

6. まとめと今後の課題

本稿では動的再構成可能ストリーミング処理エンジン (DR-SPE) のアーキテクチャとコンパイラについて検討した．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を用いることで DR-SPE を構成できることを示した．

DR-SPE は，FPGA 上に静的にストリーミング処理エンジンを構成する Streams on Wires で定義されたストリームデータ処理用の演算子と同じ動作をする回路を構成できた．XC6VLX240T-1 を用いて，提案アーキテクチャを合成，配置配線して評価した結果，Streams on Wires と同程度のスループットを実現した．DR-SPE の再構成時間は再構成の経路の通信速度が 1Mbps と低速であっても約 85 μ秒であり，再構成は十分高速に実現できる．

今後の課題は次の通りである．第一の課題は，提案アーキテクチャによる，対象アプリケーションに対する性能を評価する事である．第二の課題は，入力データに適応して，実行時にクエリを変更する動的クエリ最適化の評価を行う事である．第三の課題は，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．第四の課題はミドルウェアとしての要件を満足させることである．

[⋆1] 内部変数の初期値は含まない

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) Rene Mueller, Jens Teubner, and Gustavo Alonso. Streams on wires: a query compiler for fpgas. *Proc. VLDB Endow.*, Vol. 2, No. 1, pp. 229–240, 2009.
2) Rene Mueller, Jens Teubner, and Gustavo Alonso. Glacier: a query-to-hardware compiler. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pp. 1159–1162, New York, NY, USA, 2010. ACM.
3) Rene Mueller, Jens Teubner, and Gustavo Alonso. Data processing on fpgas. *Proc. VLDB Endow.*, Vol.2, pp. 910–921, August 2009.
4) Louis Woods, Jens Teubner, and Gustavo Alonso. Complex event detection at wire speed with fpgas. *PVLDB*, Vol.3, No.1, pp. 660–669, 2010.
5) 佐藤真琴, 田中博志, 津野田賢伸, 高田雅士, 秋田庸平, 伊藤雅樹. 再構成プロセッサ FE-GA 上への FFT のマッピング (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. 55–60, 2005-11-24.
6) Bingfeng Mei, Serge Vernalde, Diederik Verkest, and Rudy Lauwereins. Design methodology for a tightly coupled vliw/reconfigurable matrix architecture: A case study. In *Proceedings of the conference on Design, automation and test in Europe - Volume 2*, DATE '04, pp. 21224–, Washington, DC, USA, 2004. IEEE Computer Society.