# A Coarse Grain Reconfigurable Processor Architecture for Stream Processing Engine

Takefumi Miyoshi[*], Hideyuki Kawashima[†], Yuta Terada[*], Tsutomu Yoshinaga[*]
[*]The University of Electro-Communications, Japan
1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan
E-mail: miyoshi@comp.is.uec.ac.jp, terada@comp.is.uec.ac.jp, yosinaga@is.uec.ac.jp
[†]University of Tsukuba, Japan
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8577, Japan
E-mail: kawasima@cs.tsukuba.ac.jp

***Abstract*—This paper proposes a processor architecture for DR-SPE, a dynamic reconfigurable stream processing engine. DR-SPE is special-purpose hardware for stream data processing, which achieves high processing performance by exploiting parallelism in the target query. It also handles query registration and execution order of operations at runtime. Available operations in DR-SPE are the same as those in Streams on Wires. In this paper, DR-SPE is implemented on a FPGA XC6VLX240T-1, and its performance is evaluated. The results of the evaluation show that DR-SPE achieves register modification within 506 $\mu$sec when the configuration path is driven at 1 Mbps, which is not achieved by Streams on Wires. DR-SPE also achieves flexibility and can support complicated queries by providing 10$\times$10 operation units tiled onto an FPGA. DR-SPE achieves comparable operation throughput with Streams on Wires at the expense of requiring more LUTs.**

## I. Introduction

Presently, active information sources that continually generate stream data often include real-world information, for example, Twitter, Ustream, real-time stock market information, GPS data obtained by mobile equipment, surveillance cameras, and so on. Applications for anomaly detection or behavior monitoring have been developed using stream data obtained from active information sources. To ease development, the stream processing engine (SPE) technology [1] has been studied. An SPE is middleware for stream data management that vitalizes stream data as a relational stream based on the relational data model. An SPE provides relational and arithmetic operators for stream data processing. With an SPE, users are able to code desired queries over the stream data.

An SPE continually processes queries registered by users with the arrival of a stream data object. It should be noted that the generation of stream data is often very frequent. For example, the Tokyo stock exchange generates stock prices every 2 ms; the performance of a high performance IP router is 300 Tbps[2]; and industrial robots generate data for motor control every 1 ms. Obviously, it is difficult to process this type of high-rate data in real-time. Therefore, improved performance is a crucial research issue for SPE.

To improve performance, a promising approach is to exploit advanced hardware. This approach includes an implementation with network processors[3], exploitation of GPGPU or GPUTeraSort[4], improved performance of join with Cell/B.E.[5], and the implementation of special purpose hardware for stream data operations with FPGA[6][7][8][9].

Another important approach is dynamic query optimization[1][10]. Dynamic query optimization is a scheme to improve performance by dynamically changing the execution order of relational operators, such as selection, join, projection, and so on. Appropriately re-ordering operators with a quick estimation of selectivity along with data arrival improves performance significantly. To achieve the scheme, executions must be re-ordered quickly, in less than 1 ms. This is necessary because the stream data arrival rate is usually only a few milliseconds.

A novel hardware-based approach to accelerate stream processing is Streams on Wires(SoW)[6], which supports most relational and arithmetic operators. Although SoW demonstrates excellent performance on stream data processing, it is difficult to integrate with dynamic query optimization, another promising technique. The difficulty arises because of runtime cost. SoW defines primitives for executing stream data operators as libraries written in hardware description language (HDL), which Glacier[11] composes and generates for specific hardware for the target query. It is required to synthesize and place & route (P&R) in order to generate circuit information for FPGA from HDL. Before using FPGA as the specifically designed hardware, generated circuit information for FPGA is transferred to FPGA to configure the FPGA using the information. Synthesis and P&R is a heavy task and the transference of information to FPGA takes a long time. For this reason, the SoW scheme, which uses the FPGA configuration mechanism directly, cannot achieve frequent new query registration and dynamic query optimization, which needs to be shorter than 1 ms.

This paper proposes a dynamic reconfigurable processor architecture for SPE (DR-SPE), which supports fast query re-ordering for dynamic query optimization and immediate query registrations. The proposed processor architecture consists of building blocks to realize queries in a similar manner to those with SoW. To reduce redundant hardware resources, common basic building blocks are extracted for stream data processing operators. This paper presents the design and implementation of the DR-SPE and evaluates hardware resource usage and signal delay for the dynamic configuration mechanism.

The rest of this paper is organized as follows: Section II describes the motivation for the research. Section III presents the processor architecture for DR-SPE. Section IV shows the implementation and evaluation, and Section V concludes the paper.

## II. Motivation for DR-SPE

### A. Streams on Wires

Performance can be improved by implementing stream data operators as specific hardware. This is made possible through cycle-level instruction execution and using inherent parallelism. SoW[6] proposes a scheme to perform stream data operators with FPGA. FPGA is a configurable hardware device with functions modified in the device using configuration data before execution.

SoW defines primitive operators or algebras as building blocks to construct SPE on an FPGA as shown in Table I. Each algebra is defined as a circuit unit with a unified interface as shown in the table. It is possible to combine freely when the required algebras

TABLE I
ALGEBRA DEFINED AS OPERATION ELEMENTS IN STREAMS ON WIRES[6]

| | |
|---|---|
| $\pi_{a_1, ..., a_n}(q)$ | 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 $a$, $agg \in \{\mathbf{avg}, \mathbf{count}, \mathbf{max}, \mathbf{min}, \mathbf{sum}\}$ |
| $q_1\ \mathbf{grp}_{x\|c}\ q_2(x)$ | group output of $q_1$ by field $c$, then invoke $q_2$ with $x$ substituted by the group |
| $q_1\ \boxplus^t_{x\|k,l}\ q_2(x)$ | sliding window with size $k$, advance by $l$; apply $q_2$ with $x$ substituted on each wind.; $t \in \{\mathrm{time}, \mathrm{tuple}\}$: time-, or tuple-based |
| $q_1 \infty q_2$ | concatenation; position-based field join |

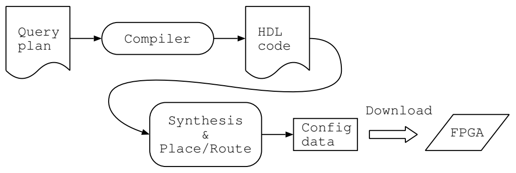


Fig. 1. Flow of generating SPE in SoW[6]

is constructed. The circuit constructed on an FPGA achieves high performance thanks to pipeline parallelism.

Figure 1 shows the flow for constructing a stream processing engine on FPGA using the scheme proposed in SoW. In SoW, a compiler referred to as a Glacier compiles a query to the HDL code that includes the instantiation of required algebras and a definition of the connection between them. The generated HDL code is compiled to the configuration data for a target FPGA with existing FPGA development tools, which provides synthesis and P&R for the HDL code. In this way, FPGA development tools are required when the new query is registered, so a long time is consumed for synthesis and P&R. On the other hand, new query registration and the dynamic query optimization that re-orders operators in a query along with arrived data must execute on the order of milliseconds. This makes it difficult to perform new query registration and dynamic query optimization in SoW.

### B. Dynamic Reconfigurable Stream Processing Engine

To solve this problem, this paper proposes a dynamic reconfigurable stream processing engine (DR-SPE). DR-SPE supports query modification, addition, and optimization without synthesis and P&R from HDL. Figure 2 shows the flow for executing a query on DR-SPE. Unlike SoW, a query is not implemented on FPGA in DR-SPE. Each query is implemented by modifying the values of registers in DR-SPE using a "dynamic configuration" path. With the flow shown in Figure 2, the HDL of DR-SPE is generated, which customizes the tuple width and operator unit bit width for the target application. Applying synthesis and P&R to the generated HDL code then generates the corresponding hardware circuit. At this time, DR-SPE cannot achieve a query. A dedicated compiler is used to compile the query for configuration data for DR-SPE. The compiled configuration data are transferred through a "dynamic configuration" at runtime.

The novel features of DR-SPE are the following:

- Perform the stream data operator,
- Achieve high computation performance by exploiting cycle-level instruction execution and inherent parallelism,

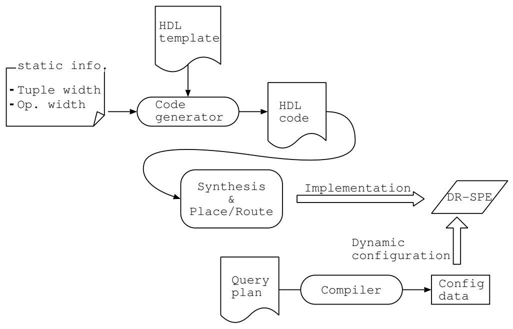


Fig. 2. Flow of generating dynamic reconfigurable stream processing engine(DR-SPE) and configuration of a query plan for it

- Modify internal module operator at runtime,
- Partially modify query and parameter configuration at runtime.

With these features, DR-SPE achieves much higher computational performance than a software implementation executable on a general purpose processor. Furthermore, these features provide the flexibility needed to enable dynamic modification of queries.

DR-SPE consists of the common execution elements needed to construct a query. A query compiler generates configuration data for implementation of the query by combining the elements.

To implement a desired operator, a logic circuit must be generated for the operator. The circuit must be mapped onto look-up tables (LUTs) and registers in the FPGA. On the other hand, DR-SPE elements are defined at the functional level. Therefore, the grain is larger than with an LUT in the FPGA. This means that the computational cost to generate configuration data and the size of configuration data are reduced significantly. Furthermore, all FPGA data are usually re-written when modifying the circuit implemented on the FPGA. Only required elements are updated when a query is configured in DR-SPE. For these reasons, DR-SPE makes dynamic reconfiguration possible.

### C. Design Issues of DR-SPE

This section describes two important design issues.

*1) Reduction of Additional Hardware Resources:* To make the DR-SPE possible to execute complicated instructions such as multiplication, division, and so on, huge hardware resources are needed to execute simple instructions such as the comparison operators like $==$ and $>$. Further, if the execution cycles for each element differ in DR-SPE, a cycle delay operator must be inserted. The amount of hardware resources and inserted delay operators is significant.

*2) Reduction of Signal Delay:* This issue is the flexibility of wiring. In the approach to generate HDL code from a query like SoW, wiring for the desired circuits is connected and incrementation/decrementation of the number of wires is modified without additional hardware cost on FPGA. On the other hand, wiring in the actual hardware (after implementing it on FPGA) is determined statically. Therefore, it is usually impossible to freely determine the connection and modify the bandwidth of the data path. To increase the options of selectable connection dynamically, a multitude of multiplexers is required, resulting in long signal delay. DR-SPE must design a processor architecture that supports sufficient connectivity without wasted hardware resources and signal delay.

## III. PROCESSOR ARCHITECTURE OF DR-SPE

To solve the issues described in Section II-C, the design of the DR-SPE processor architecture is determined by imposing the following limitations:

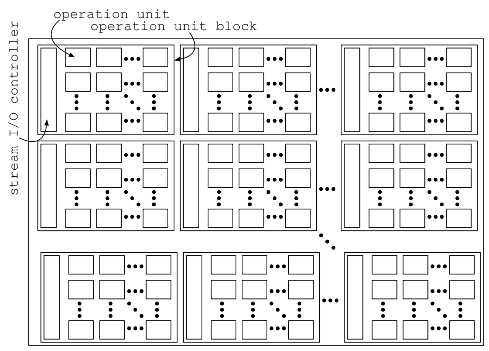


Fig. 3. Overview of DR-SPE architecture

- Constraining the execution cycles of all primitive elements to a cycle,
- Determining the bandwidth of the data path statically, and
- Reducing the configuration parameters to decrease the number of configuration registers.

In order to support equivalent functions provided by SoW, the followings are required: computation units providing algebra/logical operations and aggregations, and communication path supporting data transference between the computation units. Furthermore, in order to support Union, Grouping, and Windowing, control mechanisms for input/output data with multiple elements are required. The mechanism enables/disables the data with defined conditions. Therefore, to perform a desired query, these functions and paths must be defined as building blocks, and a mechanism to configure the building blocks must be provided in the processor architecture of DR-SPE.

Figure 3 is an overview of the DR-SPE processor architecture. DR-SPE consists of operation units, which are allocated in a tiled manner. The operation units are connected by a switch-box, which determines the connections between the neighboring units. Multiple operation units are combined to an operation unit block. A stream input/output controller exists for each operation unit block. In the operation unit block, stream input/output is controlled comprehensively. Therefore, inputs to some operation units can be defined as enabled, and other operation units in the same operation unit block can be defined as disabled.

These operation units, switch-box, and stream input/output unit have configuration registers to determine what is to be performed. DR-SPE has a mechanism to update values in these registers through "dynamic configuration" (Figure 2). With this mechanism, DR-SPE provides a dynamic configuration property.

Several research projects have studied dynamic reconfigurable processors, which achieve the desired operation using composed building blocks, such as ADRES[12] and PipeRench[13]. Since the major application of these processors is signal processing, their data path control mechanism to achieve data flow is relatively simple.

On the other hand, stream data operations require complicated data path control, such as the selection of specific data from multiple sources, enabling data under some conditions to support Union, Windowing, and Grouping. Therefore, to perform stream data operations efficiently, a stream input/output control mechanism is needed. DR-SPE has a specially designed switch-box and stream input/output controller.

## A. Operation Unit

Figure 4 shows the processor architecture of the operation unit. Input/output data for each operation unit consists of the operation target data (*tuple*), a field to save the computational result (*result*), and a flag to indicate whether the data is valid or invalid (*en*). The bit-width of *result* is a fixed size determined when this unit is synthesized. The tuple is determined along with the target input data. Therefore, the data bandwidth of the operation unit input/output is fixed statically.

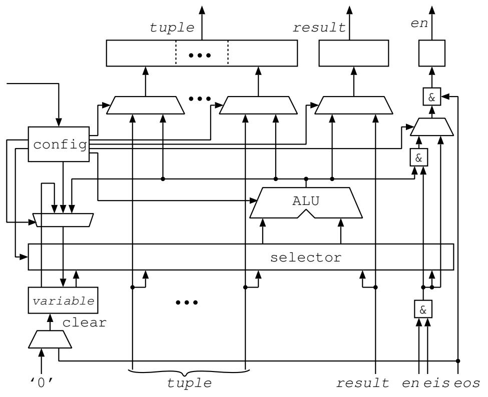


Fig. 4. Block diagram of the operation unit architecture

TABLE II
AVAILABLE OPERATIONS FOR EACH OPERATION UNIT

| type | operations |
|---|---|
| arithmetic | add, sub, +1, -1 |
| shift | 1bit left shift (with/without rotate), 1bit right shift (with/without rotate) |
| Boolean | and, or, xor, not |
| compare | $==, >, >=, !=$ |

Each operation unit has an ALU that supports basic instructions. Available instructions of the ALU are listed in Table II. The signal delay for these instructions is sufficiently short. Hence, the execution of all operation units is performed in a cycle.

Input data for the ALU are selected by a selector from the *tuple*, *result*, *en*, and *variable* as internal values. Output data from the ALU are also selectable from the *tuple*, *result*, *en*, and *variable*. Input/output data from the tuple are separated according to the operation bit-width, and each separated portion of the tuple is independently selectable.

The operation unit also has *eis* (enable input stream) and *eos* (enable output stream), which forcibly sets the input and output values as valid/invalid. When the *eis*/*eos* is '0', *en* of input/output is set to '0', which means the data are invalid.

## B. Switch-box

The switch-box manages connections between the operation units described in Section III-A. An input to the operation unit can receive output data from connected operation units through the switch-box. As shown in Figure 5, the switch-box connects an operation unit with seven surrounding operation units. The number of selectable input options is an important design issue. In this paper, the number of connection directions is arbitrarily set to seven. With the switch-box in DR-SPE, the direction of receiving input data is determined in two ways: 1) fixing a direction statically, and 2) switching two directions periodically. In case 2, the selected input is switched periodically at a time defined by the counter in the switch-box.

Figure 6 is a block diagram of the switch-box. The values of *valueA*, *valueB*, *selA*, and *selB* are configurable at runtime. The values of *selA* and *selB* correspond to the input port for the output target. The

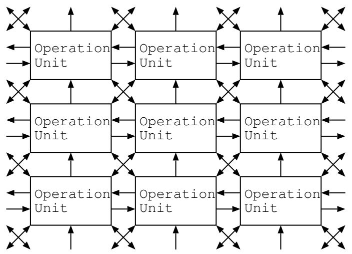


Fig. 5. Connection between operation units by switch-box

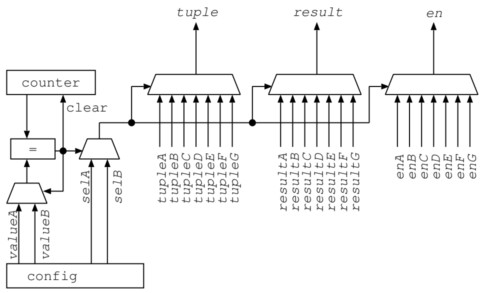

tuple
result
en
counter
clear
=
selA
selB
tupleA tupleB tupleC tupleD tupleE tupleF tupleG
resultA resultB resultC resultD resultE resultF resultG
enA enB enC enD enE enF enG
valueA
valueB
config

Fig. 6. Block diagram of switch-box architecture

values of *valueA* and *valueB* define the period to enable configured port *selA* and *selB*. When the counter equals *valueA*, the selected output port is switched from configured *selA* to configured *selB*.

## C. Stream Input/Output Controller

An operation unit performs desired operations on data, whereas a switch-box determines the connection between operation units. These two building blocks make available computations that describe simple data flow. On the other hand, summation must be performed until the timing specified for Aggregation, one of the Algebras defined in SoW. To make this possible, the output data of the operation unit that executes summation should be controlled by another operation unit, which performs the computation to determine the summation period. Furthermore, to implement Windowing and Grouping, not only each output must be controlled, but also combined input and output operation units must be controlled.

Limitation of input/output is forcibly set by *eis*/*eos* of each operation unit. Therefore, a mechanism to control *eis*/*eos* is needed. In DR-SPE, both stream input/output controller provide the mechanism. Figures 7 and 8 show the stream input controller and stream output controller. These controllers set appropriate values of *eis*/*eos* to operation units included in an operation unit block. Operation units in an operation unit block are indicated by indices.

In the stream input controller, output data for *eis* are selected from four candidates: demultiplexed counter value, demultiplexed value of content-addressable memory(CAM), *result* of input data, and '1'. Note that *eis* for the operation units that have equal or larger index values than the counter value must be set to '1'. Therefore, when the counter value is zero, valid/invalid flag of input data is always equal to the value of *en* of the input data. It is also possible to invert the output *eis* value. It is possible to implement CAM that is readable in a cycle by LUTs and registers in FPGA, however, the implementation cost is large. Therefore, the number of CAMs is limited when tiling many operation units in an FPGA.

In the stream output controller, output data for *eos* are selected from three candidates: demultiplexed counter value, *result* of input

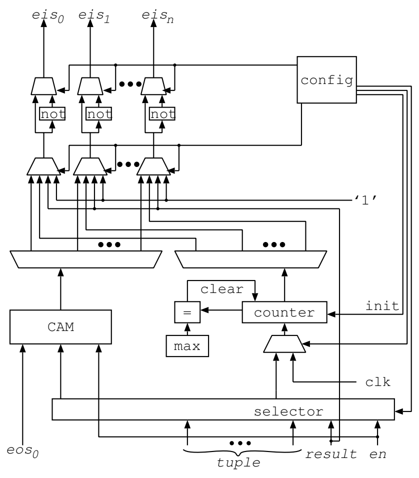


Fig. 7. Block diagram of stream input controller

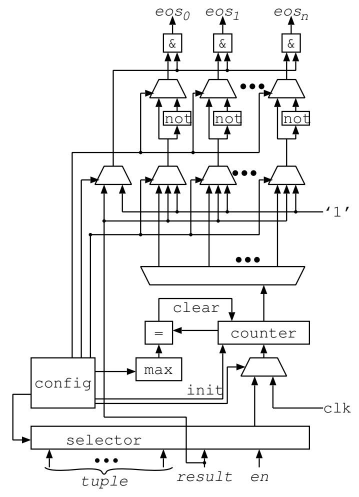


Fig. 8. Block diagram of stream output controller

data, and '1'. Just as with the control of *eis*, *eos* in the operation units that have equal or larger index values than the counter value must be set to '1'. It is also possible to invert the output *eos* value.

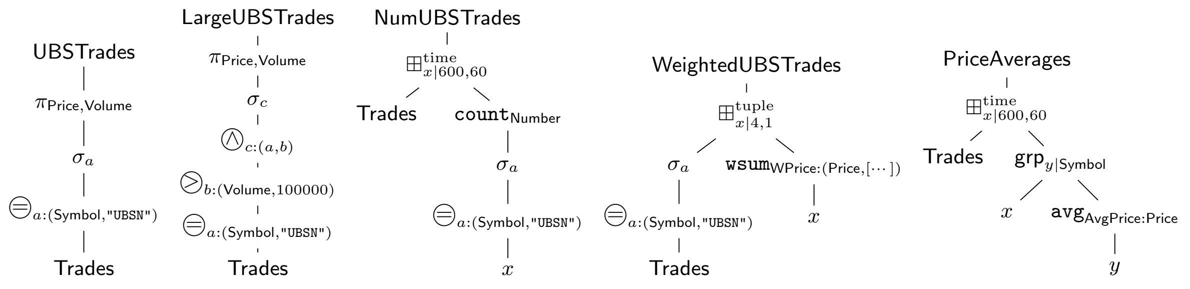
## IV. Implementation and Evaluation

This section evaluates the proposed DR-SPE. First, the implementation of the proposed DR-SPE is shown in Section IV-A. Second, the evaluation results for the DR-SPE are shown in Section IV-B. The evaluation includes configuration time, throughput, scalability, and additional hardware resources.

## A. Implementation of the DR-SPE

The proposed DR-SPE was implemented on a Xilinx FPGA XC6VLX240T-1. The Virtex®-6 FPGA ML605 Evaluation Kit is used. The specifications of the FPGA are shown in Table III. Xilinx ISE 12.1 Logic Edition is used as an FPGA development tool, and XST compiler is used for synthesis.

In this paper, the proposed DR-SPE is evaluated by comparison of the implementation results for queries Q1-Q4 shown in SoW[6] (Figure 9) with DR-SPE and SoW. It should be noted that Q5 is out of the scope of this paper, so that it is not implemented or compared. Since the purpose of this section is to evaluate hardware


UBSTrades
$\pi_{\text{Price,Volume}}$
$\sigma_a$
$\ominus_{a:(\text{Symbol},\text{"UBSN"})}$
Trades
LargeUBSTrades
$\pi_{\text{Price,Volume}}$
$\sigma_c$
$\wedge_{c:(a,b)}$
$>_{b:(\text{Volume},100000)}$
$\ominus_{a:(\text{Symbol},\text{"UBSN"})}$
Trades
NumUBSTrades
$\boxplus^{\text{time}}_{x|600,60}$
Trades
$\texttt{count}_{\text{Number}}$
$\sigma_a$
$\ominus_{a:(\text{Symbol},\text{"UBSN"})}$
$x$
WeightedUBSTrades
$\boxplus^{\text{tuple}}_{x|4,1}$
$\sigma_a$
$\texttt{wsum}_{\text{WPrice:(Price},[\cdots])}$
$\ominus_{a:(\text{Symbol},\text{"UBSN"})}$
$x$
Trades
PriceAverages
$\boxplus^{\text{time}}_{x|600,60}$
Trades
$\texttt{grp}_{y|\text{Symbol}}$
$x$
$\texttt{avg}_{\text{AvgPrice:Price}}$
$y$

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

Fig. 9. Sample queries listed in SoW as example (Cited from [6])

TABLE III
SPECIFICATIONS OF XC6VLX240T-1

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

TABLE IV
PARAMETERS OF A DR-SPE IMPLEMENTED FOR THE EVALUATION

| | |
|---|---|
| Tuple bit width | 96-bit |
| Operator bit width | 32-bit |
| #. of units in a block | 8 |
| Available ways for union | 8 |
| #. of configuration registers in a block | 506-bit |

TABLE V
RESULT OF RESOURCE USAGE FOR AN OPERATION UNIT, A SWITCH-BOX, AND AN INPUT/OUTPUT CONTROLLER

| module | #. of Slice Registers | #. of Slice LUTs |
|---|---|---|
| An operation unit | 181 | 483 |
| A switch-box | 22 | 285 |
| An input/output controller | 24 | 74 |

TABLE VI
RESULT OF RESOURCE USAGE FOR DR-SPE INCLUDING 10×10 OPERATION UNITS

| | resource usages | ratio |
|---|---|---|
| #. of Slice Registers | 20038 | 6 % |
| #. of Slice LUTs | 88421 | 56 % |

resource usage and signal delay to implement a DR-SPE that includes additional hardware resources over a simple static stream processing engine. Q5 cannot be implemented because it requires hardware resource usage and signal delay for implementing CAM that depends strongly on the address width and data width. The additional hardware resources and signal delays are relatively large when a small CAM is implemented, or the additional hardware resources and signal delays are relatively small when a large CAM is implemented. Therefore, for the evaluation to be under fair conditions, Q5 is omitted from this evaluation.

The evaluation target assumes a DR-SPE with the parameter set shown in Table IV. The parameter set is assigned to perform queries described in SoW. In the SoW, the input stream is a type of market and the tuple consists of *symbol*, *time*, and *price*. When the *symbol* is a 4-byte (= 32-bit) string such as "UBSN", and each of *time* and *price* are presented in 32-bit, the tuple bit width is 96-bit. Operator bit-width of the ALU in the operation unit is 32-bit, which enables instructions to execute naturally. To perform Q4 (in Figure 9), a 5-way union is required. For a 5-way union using stream input/output controller, the number of units in a block is set to 8. This setting allows for an 8-way union, and so Q4 is satisfied. Due to these parameters, the number of configuration registers in a block becomes 506-bit.

Hardware resource usage for the designed operation unit, switch-box and stream input/output controller are shown in Table V. The maximum executable frequency of each unit is 232.8MHz, 430.3MHz, and 440.3MHz, respectively. Note that the stream input controller does not include CAM.

## B. Evaluation of DR-SPE

This section evaluates the proposed DR-SPE in terms of the configuration time, the flexibility, the operation throughput, and the additional hardware resource usages for the reconfiguration mechanism.

*1) Configuration time:* For the parameters shown in Table IV, the configuration registers for a block become 506-bit. It should be noted that even if the implementation of the "dynamic configuration" path (Figure 2) is driven at 1 Mbps, the proposed architecture modifies the registers within 506 $\mu$sec. This implies that a re-ordering of the stream relation operators is achieved within 506 $\mu$sec.

In the case of SoW, reconfiguration of the entire FPGA is required for re-ordering the stream relational operators, which takes about 20.5-21.7 seconds in the given environment[1]. Therefore, our proposed architecture supports quick configuration for dynamic query optimization, which is not achieved by SoW.

*2) Flexibility:* The flexibility of DR-SPE is an important feature to realize user-desired queries quickly. Even though DR-SPE requires additional hardware resources to provide a reconfigurable mechanism over an implementation of a simple stream processing engine, it is possible to tile many operation units in an FPGA. Table VI shows hardware resource usage of a DR-SPE tiled with 10×10 operation units. The maximum executable frequency of 10×10 DR-SPE is 172.1 MHz. This result shows that a query, which consists of up to 100 primitive operations, can be processed on the proposed DR-SPE implemented on XC6VLX240-T. Therefore, it is obvious that DR-

[1]CPU: Core2Quad 2.66GHz, Memory: 4GB, OS: Windows XP(32-bit)

TABLE VII
RESULT OF RESOURCE USAGE FOR STATIC QUERIES USING ALGEBRA

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

TABLE VIII
NUMBER OF REQUIRED OPERATION UNITS FOR EACH QUERY

| Query | #. of units | #. of Slice Registers | #.of LUTs |
|---|---|---|---|
| Q1 | 2 | 362 | 966 |
| Q2 | 4 | 724 | 1032 |
| Q3 | 11 | 1991 | 5313 |
| Q4 | 15 | 2715 | 7254 |

SPE can process each of Q1-Q4 which consist of operations lower than 100.

*3) Throughput:* This section compares the throughput for query operations of SoW and DR-SPE. In the implementation of Q1-Q4 by XC6VLX240T-1 with SoW, the maximum executable frequency for all queries is about 200MHz. The maximum executable frequency for DR-SPE is about 172.1MHz, so that the frequency is less than with SoW. The operation throughput of both SoW and DR-SPE is up to 1 tuple/cycle. Therefore, when a tuple bit-width is 96 bits, the maximum operation throughput of SoW and DR-SPE is about 19,200 Mbps and 16,521 Mbps, respectively. Both maximum operation throughputs are over a data arrival rate of 1,000,400 packets/s (available at the laboratory[6]). Therefore, the operation performance of the proposed DR-SPE and SoW can be regarded as comparable at the viewpoint of practical system operation.

It should be noted that the maximum executable frequency depends on the signal delay of the critical path in the circuits. It is possible to change hardware circuits when SoW is used. However, doing so may decrease the maximum executable frequency. On the other hand, in DR-SPE, the hardware circuit is not changed to implement queries, and so performance degradation of the maximum executable frequency and the maximum operation throughput does not occur. This contributes to the novel flexibility of DR-SPE, because implementation of desired queries is easily achieved without increasing signal delays.

*4) Additional resource usage:* Required hardware resources are shown in Table VII for Q1-Q4 shown in Figure 9 with SoW. When the same Q1-Q4 are implemented with DR-SPE, required hardware resources are shown in Table VIII. As shown in Table VII and Table VIII, register usage of DR-SPE is about 1.8-3.9 times larger than with SoW, and LUT usage of DR-SPE is about 19.5-120.8 times larger than with SoW. To implement multiplexers for a dynamic reconfiguration mechanism of DR-SPE, many more LUTs are required. On the other hand, the number of increased registers is much less than the number of increased LUTs. This is because wasted registers for a stored temporal result bit and for stored configuration data increase register usage of DR-SPE.

## V. CONCLUSIONS

This paper proposed a DR-SPE. It provides both high computation performance for stream data operations and a dynamic reconfiguration mechanism to sufficiently support dynamic query optimization. DR-SPE consists of operator units, switch-box and stream input/output controllers, which support operators equivalent to Streams on Wires(SoW)[6].

The proposed DR-SPE was implemented on XC6VLX240T-1 and evaluated in terms of configuration time, flexibility, operation throughput, and additional hardware resource usage for reconfiguration mechanism. As for reconfiguration performance, the architecture achieved register modification within 506 $\mu$sec when the configuration path was driven at 1 Mbps, which was not achieved by SoW. The proposed DR-SPE also achieves flexibility to support complicated queries by providing $10 \times 10$ operation units tiled onto an FPGA. The signal delay is slightly larger with DR-SPE than with SoW.

In future work, we will focus on optimizing the processor architecture of DR-SPE and implement a query compiler for it. Furthermore, we will implement dynamic query optimization algorithms on it. Then we plan to apply DR-SPE to actual packet streaming applications.

## ACKNOWLEDGMENT

This work is partially supported by "KAKENHI (#22700090)", "KAKENHI (#23700054)", and "Early-concept Grants for Exploratory Research on New-generation Network".

## REFERENCES

[1] Ron Avnur and Joseph M. Hellerstein. Eddies: continuously adaptive query processing. *SIGMOD Rec.*, 29:261–272, May 2000.

[2] Cisco Carrier Routing System. http://www.cisco.com/en/US/products/ps5763/index.html.

[3] Brian Gold, Anastassia Ailamaki, Larry Huston, and Babak Falsafi. Accelerating database operators using a network processor. In *Proceedings of the 1st international workshop on Data management on new hardware*, DaMoN '05, New York, NY, USA, 2005. ACM.

[4] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 325–336, New York, NY, USA, 2006. ACM.

[5] Buğra Gedik, Philip S. Yu, and Rajesh R. Bordawekar. Executing stream joins on the cell processor. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 363–374. VLDB Endowment, 2007.

[6] Rene Mueller, Jens Teubner, and Gustavo Alonso. Streams on wires: a query compiler for FPGAs. *Proc. VLDB Endow.*, 2(1):229–240, 2009.

[7] Rene Mueller, Jens Teubner, and Gustavo Alonso. Data processing on FPGAs. *Proc. VLDB Endow.*, 2:910–921, August 2009.

[8] Louis Woods, Jens Teubner, and Gustavo Alonso. Complex Event Detection at Wire Speed with FPGAs. *PVLDB*, 3(1):660–669, 2010.

[9] Jens Teubner and Rene Mueller. How soccer players would do stream joins. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 625–636, New York, NY, USA, 2011. ACM.

[10] Kwanchai Eurviriyanukul, Norman W. Paton, Alvaro A. A. Fernandes, and Steven J. Lynden. Adaptive join processing in pipelined plans. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 183–194, New York, NY, USA, 2010. ACM.

[11] 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*, pages 1159–1162, New York, NY, USA, 2010. ACM.

[12] 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, pages 21224–, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Yuan Chou, Pazhani Pillai, Herman Schmit, and John Paul Shen. Piperench implementation of the instruction path coprocessor. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, MICRO 33, pages 147–158, New York, NY, USA, 2000. ACM.