J. Shanghai Jiao Tong Univ. (Sci.), 2017, 22(4): 493-503
DOI: 10.1007/s12204-017-1862-0

Design and Implementation of a Data-Driven Dynamical Reconfigurable Cell Array

SHAN Rui[1*] (山　蕊), *LI Tao*[2] (李　涛), *JIANG Lin*[3] (蒋　林)
DENG Junyong[3] (邓军勇), *SHEN Xubang*[1] (沈绪榜)
(1. School of Microelectronics, Xidian University, Xi'an 710071, China; 2. Research Center for ASIC Design, Xi'an University of Posts and Telecommunications, Xi'an 710121, China; 3. School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

Abstract: The nature of dataflow computation demands the heavy flow of tokens amongst computation nodes. Traditional reduced instruction-set computer (RISC) processors are not suitable for such style computation. Devices that use long wire buses are not suitable for dataflow either. Reconfigurable computing devices (RCDs) consist of data transfer wires and computing resources. With minor modifications, reconfigurable cells can be adopted to perform dataflow computation. A reconfigurable cell array (RCA) is presented in this paper and it is suitable for dataflow computation. This cell array has a dynamic reconfigurable storage model. The distinctive features of the architecture include dataflow reconfigurable cells and reconfigurable storage. Dataflow applications can be mapped easily and effectively onto the cells. Reconfigurable storage is mainly used to manage data access and transmission. Furthermore, computation and data management are separated. Meanwhile, dynamical reconfiguration is accomplished, when some clusters of cells work in configuration mode and other clusters work in computation mode. The dataflow graphs of some algorithms are mapped onto our architecture, and the performance results are compared with those of CPU and GPU.
Key words: reconfigurable architecture, cell array, dataflow computing, storage structure, distributed storage
CLC number: TP 391　　**Document code:** A

0 Introduction

In the field of high-performance computing, reconfigurable computing devices (RCDs) (such as field-programmable gate arrays (FPGAs)) are gaining more widespread interest[1]. Many reconfigurable architectures have been proposed in the literature. Miyamori and Olukotun[2] discussed reconfigurable multi-media array coprocessor (REMARC) which consists of a global control unit and an 8×8 array of nano-processors. The array of nano-processors is similar to a reduced instruction-set computer (RSIC) processor, and it includes an arithmetic logical unit (ALU), an instruction random access memory (RAM), a data RAM, a register file, and some data input and output registers. A global program counter (PC) is used to fetch instructions for all of nano-processors in the 8×8 array. The array may operate in either multiple instruction multiple data (MIMD) or single instruction multiple data (SIMD) model. Data can be transferred via adjacent neighbor connections or via line/row bus. Singh et al.[3] proposed MorphoSys system which consists of tiny-RISC processors, reconfigurable cell array (RCA), frame memory and direct memory access (DMA) controller. Reconfigurable cell comprises a shift unit, an ALU-multiplier, two multiplexers at the reconfigurable cell inputs, an output register, a feedback register and a register file. Data transformation in RCA is via neighbor interconnections and long wires. Data communication between RCA and outside uses the frame memory and DMA. It works in SIMD mode. Veredas et al.[4] proposed architecture for dynamically reconfigurable embedded system (ADRES) which is a very long instruction word (VLIW) architecture with a reconfigurable processing element (PE) array. Data communication between PE array and outside is achieved by a multi-port register file, so data transfer bandwidth is limited. A fully programmable coarse-grain reconfigurable processor was

Received date: 2016-10-10
Foundation item: the National Natural Science Foundation of China (Nos. 61136002, 61272120, 61634004 and 61602377), the Shaanxi Provincial Co-ordination Innovation Project of Science and Technology (No. 2016KTZDGY02-04-02), the Shaanxi Provincial Science and Technology Research Fund (Nos. 2013KTZB01-07, 2014ZS-08 and S2015TQGY0166), and the Fund of Shaanxi Education Bureau (No. 2050205)
∗**E-mail:** shanrui0112@163.com

proposed in 2006[5]. The commercial processor contains three types of processing array elements (PAEs): ALU-PAEs, function-PAEs and RAM/IO-PAEs. ALU-PAEs are used for computing. Function-PAEs are used for application controlling. RAM/IO-PAEs are used for data transfer. Zhu et al.[6] presented a reconfigurable multimedia system (REMUS) architecture which consists of an ARM9, two RCAs, an entropy decoder and some auxiliary modules. RCAs are mainly used to perform the media applications of data parallel part. Reconfigurable cell comprises two input data selection units, an ALU, an output data selection, a temporary register and a result register. Data transfer in RCAs is mainly through the register array. Li et al.[7] discussed a multi-level parallel polymorphic array architecture for graphics (PAAG) architecture which consists of several levels of clusters of processors and seamlessly integrates data parallelism, thread parallelism, operation parallelism (found in FPGA and application specific integrated circuit (ASIC) designs), and distributed instruction parallelism. A cluster is a 2D array of processors. The processor is a two-way instruction parallel unit. Data transfer between adjacent processors is completed through neighbor connections as well as data routers.

From above reconfigurable architecture, we can see that an RCA can be used as a functional component of a processor, a coprocessor, or an attached processor[8]. Modern coarse grain reconfigurable cells often use RISC processors or simplified RISC processors as computing nodes[9]. Some RCAs use shared registers to exchange data between reconfigurable cells while others use neighboring connections to transfer data between adjacent reconfigurable cells. For long distance data transformation, line bus and row bus are used. However, buses have long delays.

Dataflow application is represented by a dataflow graph which can be mapped onto RCA[10]. The execution of dataflow graph often needs to move a large quantity of data among adjacent nodes. Such intensive data movements cannot be satisfied by RISC processor-based architectures.

A data-driven dynamical RCA (DD-RCA) is proposed in this paper. In DD-RCA, a dual-rail protocol (REQ/ACK) is used for data exchange to mimic dataflow computation. An operation with a data transmission typically takes one cycle. This greatly improves data transmission efficiency. A reconfigurable storage is also proposed in this paper. This storage takes over the data transmission control task, so the nodes can focus on the computing task.

1 DD-RCA Architecture

DD-RCA can be used as a functional component of a system or as a coprocessor. Its architecture is shown in Fig. 1. It consists of three major parts: a cell cluster, a reconfigurable switch, and a reconfigurable distributed memory controller. This architecture is scalable. It is a system of 4×4 clusters.

Each cluster is a 4×4 cell array. A cell is connected with up to four neighbor cells, as shown in Fig. 2. The cluster is mainly used for computing. An application is represented by a dataflow graph. A node in a dataflow diagram is usually mapped onto a computing cell, and a dataflow graph can be directly mapped onto some clusters.

In this paper, we present the design of a dataflow reconfigurable cell. For a computation node, input data may come from one or more of the neighbors or from outside of the cluster. A dataflow cell is fired for

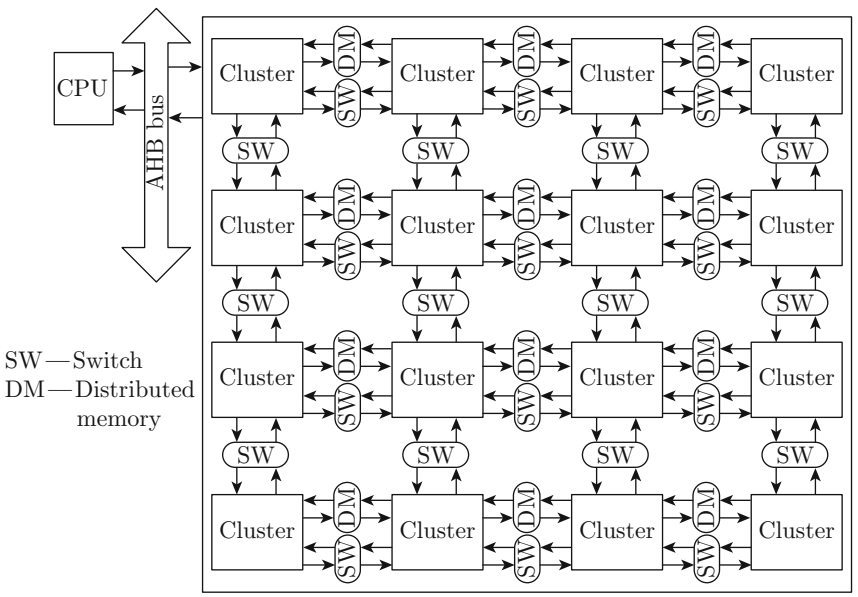


Fig. 1 Architecture of DD-RCA

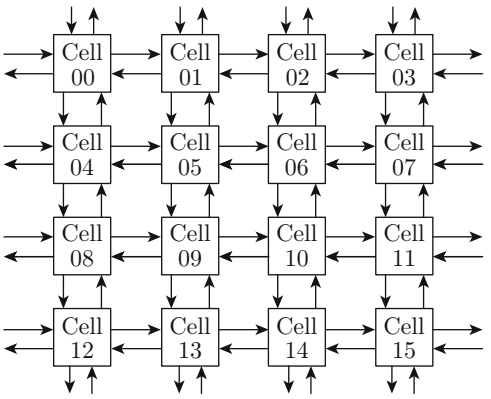


Fig. 2 Cluster structure

execution only when all of the input data arrive. The cell can continue execution when its output is consumed by another cell. This corresponds to a single buffer static dataflow computation.

Reconfigurable switches route data between adjacent clusters. Each switch connects an edge of cells of one cluster with an edge of cells of another cluster, so one edge of a cluster can exchange data with an edge of another cluster.

When a cluster needs to exchange a large quantity of data with another (such as the case in openVX filing), it uses one of the distributed memories. Each reconfigurable distributed memory is attached to a controller.

2 Configurable Mechanism

DD-RCA is dynamically reconfigurable. A part of it can execute normal operations, while another part set in configuration mode can receive configuration information to reconfigure itself. An H-tree network[11], as shown in Fig. 3, is used to transmit configuration information. Reconfiguration information can be passed along the H-tree network to any part that needs reconfiguring.

DD-RCA supports two modes of operation: working mode and configuration mode. Mode switching information is also dispatched along the H-tree network, either in a single direction or in several directions. Upon initialization, all clusters/switches are set in configuration model. When the configuration process is finished, clusters can be set in working mode. When some clusters are performing computation, other clusters including configurable switches and configurable distributed memory need reconfiguration.

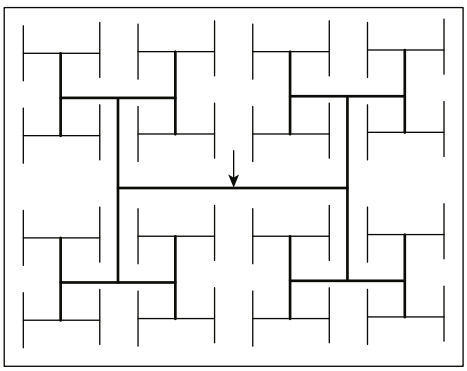


Fig. 3 Configuration dispatch network

3 Reconfigurable Components

DD-RCA architecture has three basis reconfigurable components: dataflow reconfigurable cell, reconfigurable switch, and reconfigurable distributed memory.

3.1 Dataflow Reconfigurable Cell

Dataflow reconfigurable cell is mainly used for computing. It can be configured as only executing one operation or continually executing a sequence of operations. After executing the last operation in a sequence, it restarts from the first operation. Dataflow reconfigurable cell is a fine grained device and it only supports some simple operations, as shown in Fig. 4. Operand A comes from one of the four neighbors, or a memory cell. Operand B comes from one of the four neighbors, a memory cell, or an immediate data. Dataflow reconfigurable cell supports 16 bit immediate operand and it contains eight data registers. It does not have data RAM.

A + B	A − B	A & B
A | B	A ^ B	∼ A
A ≫ B	A ≪ B	A > B
A < B	A==B	abs(A)

Fig. 4 Dataflow reconfigurable cell supporting operation

The structure of dataflow reconfigurable cell is shown in Fig. 5. It has a mode register, a configuration information unit, a configure point counter and a dataflow computing cell.

The mode register indicates whether the dataflow reconfigurable cell is in working mode or configuration mode. This register saves the mode information from the H-tree network.

The configuration information unit includes sixteen configuration registers, each of which stores an operation. Configure point is set to zero after switching to configuration mode. Configuration information that comes from the H-tree network is incrementally written into sixteen registers (denoted as R0 to R15).

The configuration point counter is analogous to a program counter. It points to the next operation to be performed. After an operation is finished, this counter is increased by one and then is restarted when the last operation in the sequence is executed.

The dataflow cell is mainly used for computing. The cell has input multiplexers, an ALU unit and a fan out

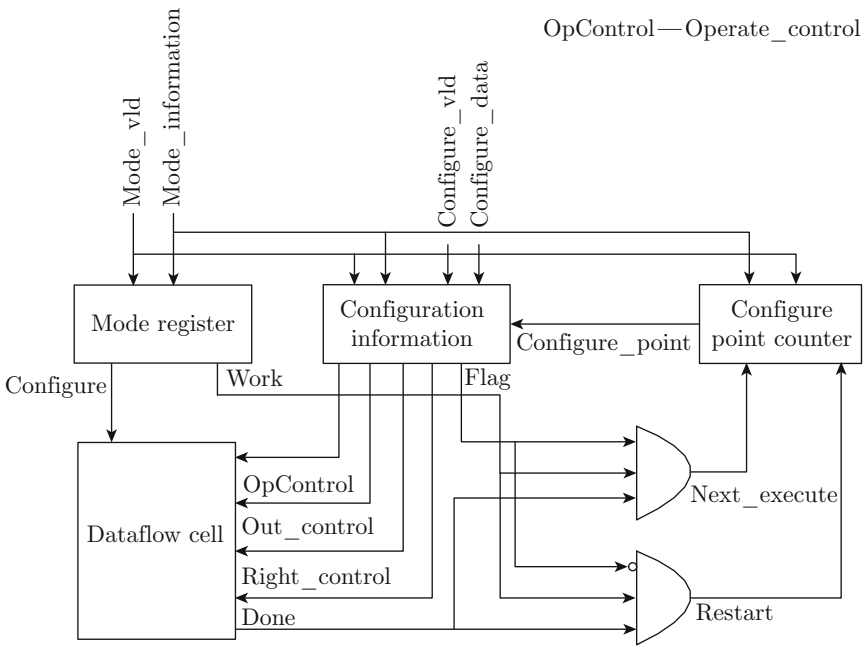


Fig. 5 Dataflow reconfigurable cell structure

unit, as shown in Fig. 6. The information stored in each configuration register includes the opcode, the directions of the input operands (specified by the multiplexer selections), and the output directions (fan outs).

The dual-rail hand-shaking is used for pipelining. For each input/output direction, a pair of {valid, ready} signals accompany the data signal. When all desired inputs of an operation arrive, the cell is fired to execute the operation. The result is sent out to the fan-out unit which waits until all outputs are received by other cells.

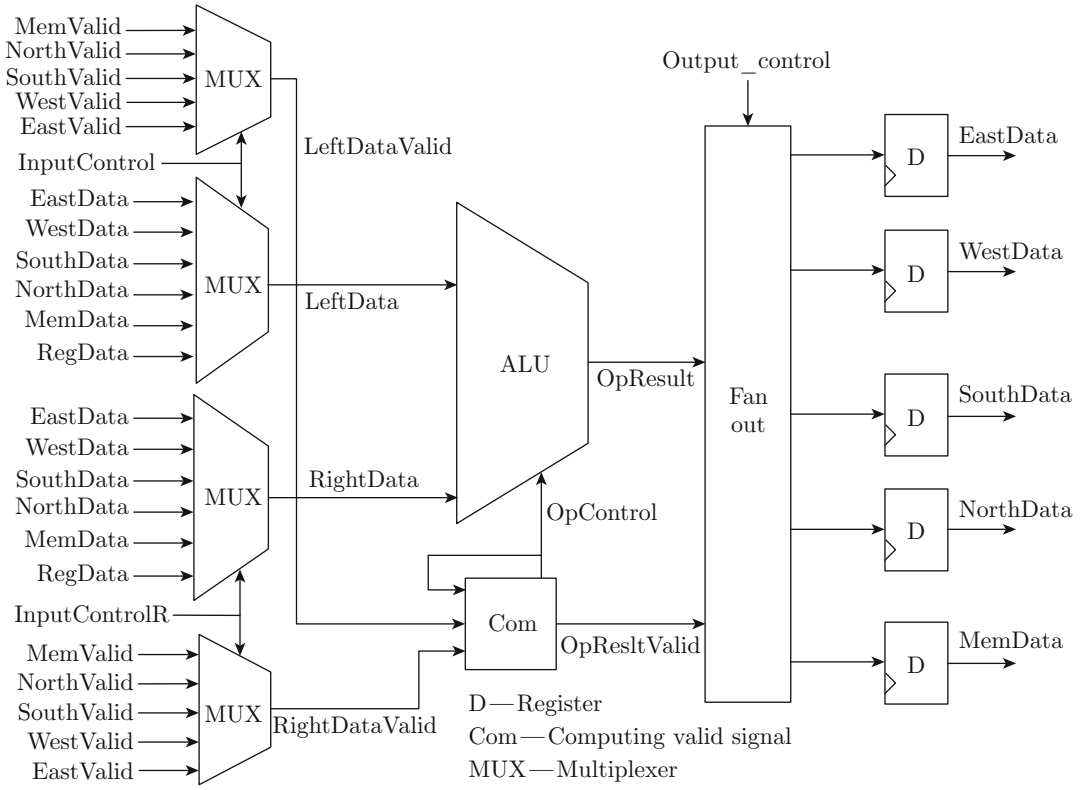


Fig. 6 Dataflow cell architecture

3.2 Reconfigurable Switch and Distributed Memory

The reconfigurable switch helps data exchange between two adjacent clusters. It consists of one configuration register and eight multiplexers, as shown in Fig. 7.

The multiplexers in the switch must also observe the dual-rail hand-shaking protocol. Its architecture is shown in Fig. 8. Data can be sent to an adjacent cluster only when the destination port is ready. A ready signal is not back propagated to the input side until all fan outs are ready.

A reconfigurable distributed memory is shared by two horizontally adjacent clusters. The two clusters can exchange information using the memory when the amount of information is large. Essentially, it is an on-chip static random access memory (SRAM) used to buffer intermediate results (such as the openVX tiling data). As shown in Fig. 9, it is composed of a configuration interface, a micro-code memory, a data memory, a reconfiguration controller, four input register units and four output register units.

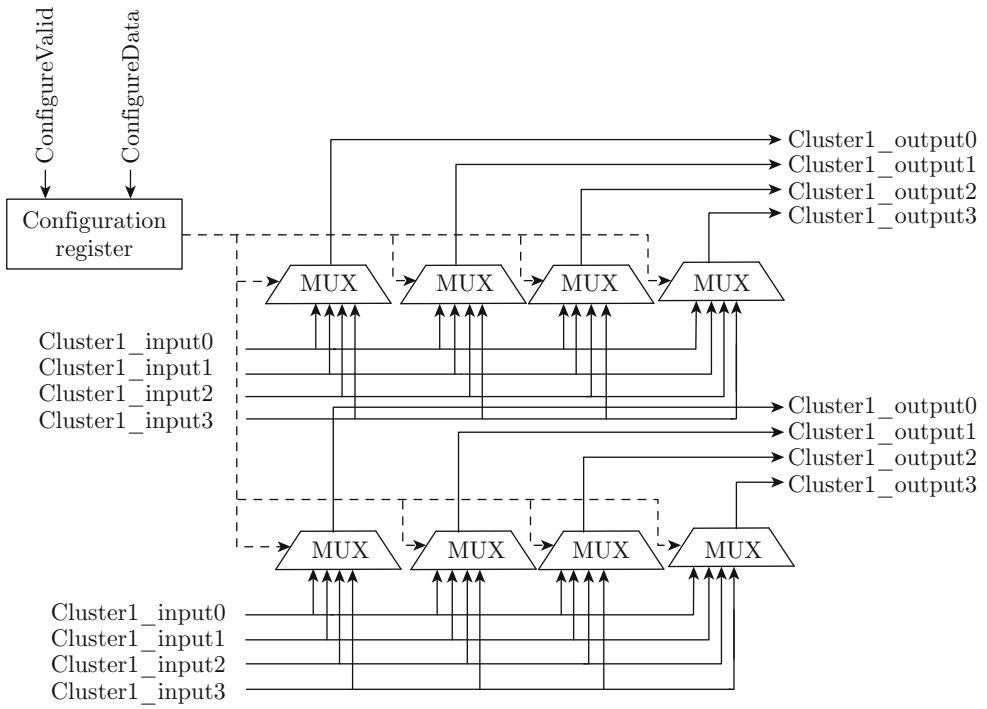


Fig. 7 Reconfigurable switcher architecture

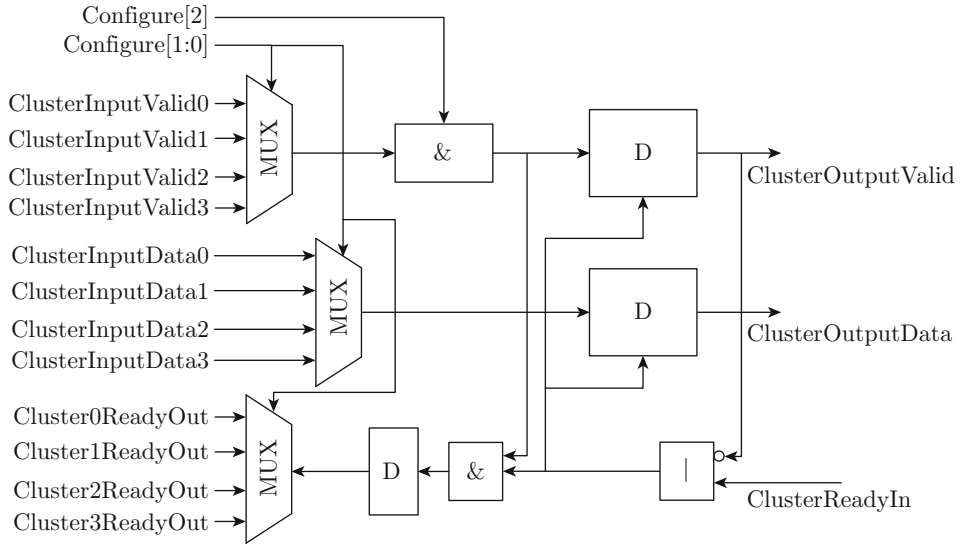


Fig. 8 Multiplex unit architecture

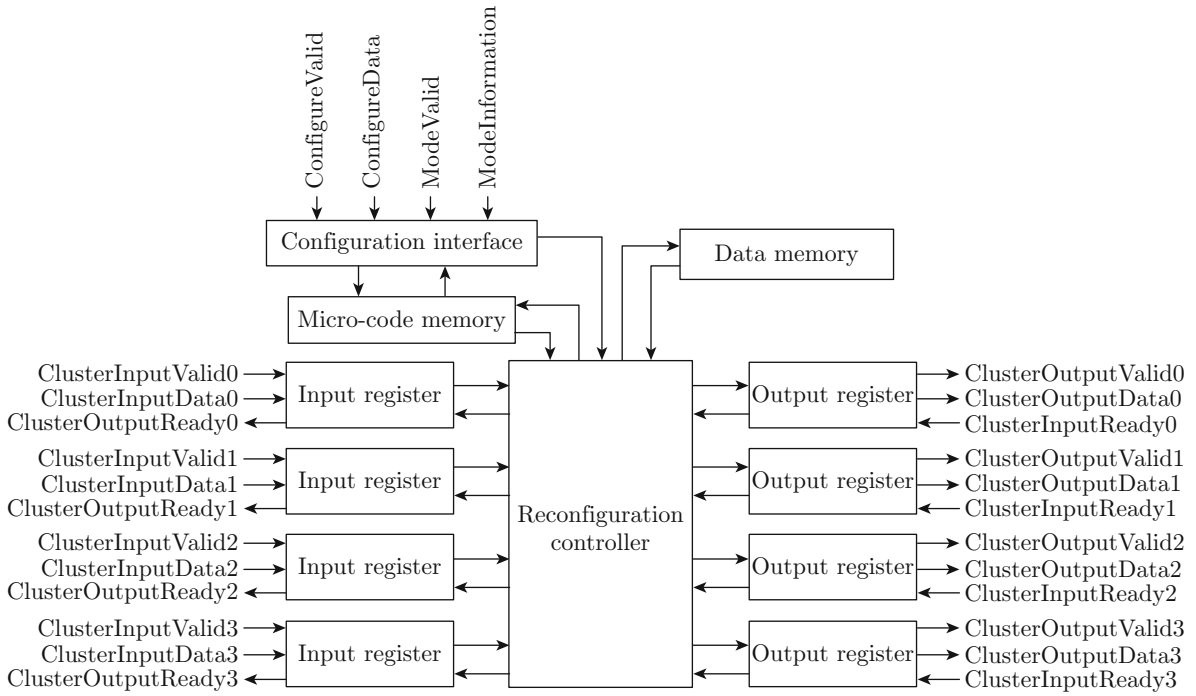


Fig. 9 Reconfigurable distributed memory structure

Configuration interface takes charge of writing configuration information to micro-code memory. There is also a mode register in configuration interface for saving mode information.

Micro-code memory is used for saving configuration information or micro-code. Data memory is used for saving data from external storage or buffering intermediate results. It can be seen as an RAM or first in first out (FIFO).

Input register units can receive data from cluster and provide the data to reconfiguration controller. Output register units can receive data from reconfiguration controller and provide the data to cluster. Their realizations are similar to multiplex units.

Reconfiguration controller is in charge of managing data. It decides where data come from and where they go and communicates with two adjacent clusters through four input register units and four output register units. Reconfiguration controller is much like an RISC processor with some special instructions (such as receive (REC) and write (WRT)). Instruction REC is used to receive data from four input register units. Instruction WRT is used to send data to four output register units.

4 Implementation of Various Tasks

DD-RCA employs dataflow computing units, data exchange switches, and shared distributed memories. It is suitable for digital signal processing, computer vision computations, and so on. To study the effectiveness of DD-RCA architecture, we implement a simple computer vision application taken from the openVX specification[12] and a cascaded integrator-comb (CIC) filter[13] on DD-RCA.

4.1 Computer Vision Application

An example taken from the openVX specification 1.0 is shown in Fig. 10. It is an openVX graph with three nodes.

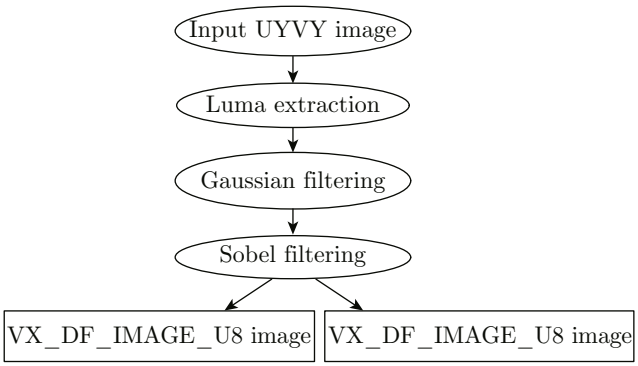


Fig. 10 A computer vision application

It takes a UYVY format image as input. The first node extracts the luma information (Y component) from the image. Gaussian filtering is performed on the luma image to smooth the image. In the next step, Sobel filtering is used to generate X edges in both the x and y directions.

The luma extraction node separates the Y-components from a UYVY format image, i.e. it converts U0Y0V0Y1 and U2Y2V2Y3 to Y0Y1Y2Y3. This can be expressed by a dataflow graph, as shown in Fig. 11.

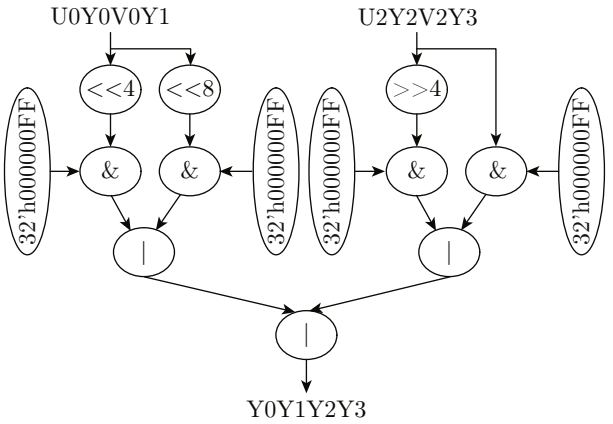


Fig. 11 Dataflow extraction diagram

ANDing U0Y0V0Y1 with x00FF0000 and x000000FF respectively results in x00Y00000 and x000000Y1. ANDing U2Y2V2Y3 with x00FF0000 and x000000FF respectively results in x00Y20000 and x000000Y3. Shifting left x00Y00000 by 8 and left x000000Y1 by 16 and then ORing them together result in xY0Y10000. Shifting right x00Y20000 by 8 and then ORing with x000000Y3 result in x0000Y2Y3 which is then ORed with xY0Y10000 to produce xY0Y1Y2Y3. These operations can be performed in parallel.

Gaussian filtering is mainly used to smooth images[12]. Here, the convolution coefficients are given as

$$h(n) = \{1, 2, 1; 2, 4, 2; 1, 2, 1\}\frac{1}{16}, \tag{1}$$

The convolution process is expressed by

$$\begin{aligned} y(n) = & x(n) * h(n) = \\ & x(0)y(0) + x(1)y(1) + \cdots + x(8)y(8), \end{aligned} \tag{2}$$

where $x(n)$ is the input pixel point information, and $y(n)$ is the result after Gaussian filtering. Gaussian filtering dataflow diagram is shown in Fig. 12. Since the coefficients in Eq. (1) are all multiples of 2, the multiplications can be replaced by shifts. Division by 16 can be replaced by a right shift of 4, as shown in Fig. 12.

Sobel filtering[14] is commonly used in edge detection, and it is usually performed along the horizontal and vertical directions respectively. Sobel filtering matrices for the x and y directions are shown as

$$\boldsymbol{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{3}$$

Note that the coefficients are also powers of 2. This computation process is similar to Gaussian filtering. But the convolution coefficients are different. The dataflow diagram is also similar and is shown in Fig. 13.

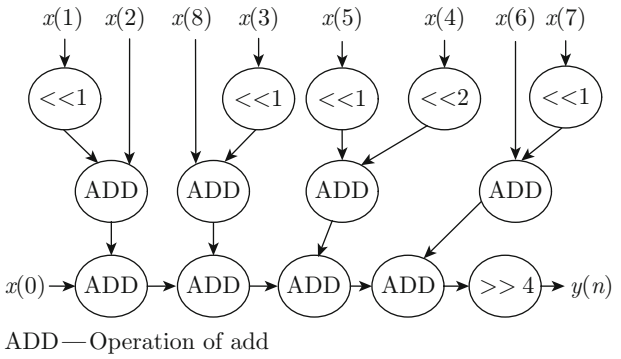


Fig. 12 Gaussian filtering dataflow diagram

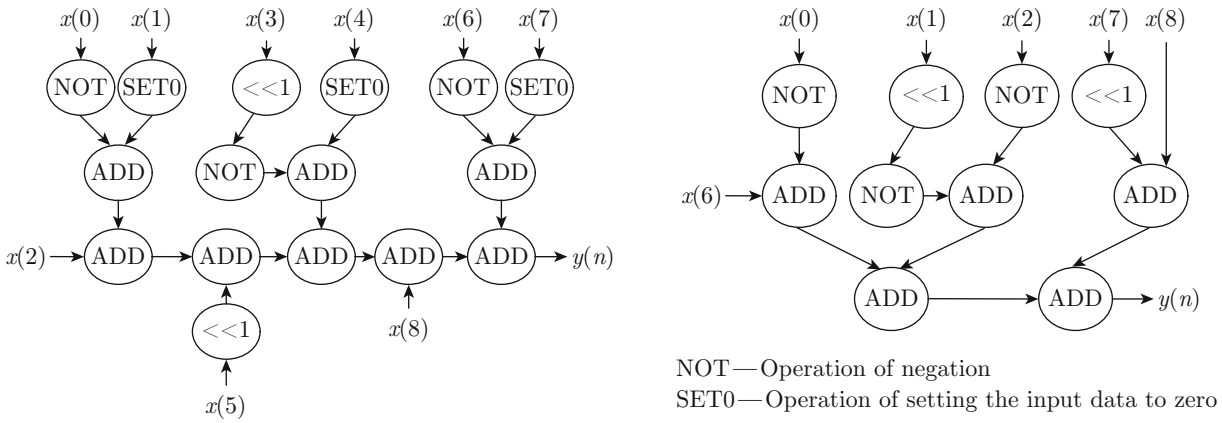
We can easily map the luma extraction, Gaussian filtering and Soble filtering onto DD-RCA. The mapping is shown in Fig. 14, where dotted lines stand for connections with distributed data memory.

Data exchange between Gaussian filtering node and extraction node is through the data distributed memory. Part of configuration information is described as follows. In the configuration information, ADDi stands for adding with immediate data, REC R1-RL3 stands for receiving the data from the left input 3 to Register 1, ST R2-R1 stands for storing the data of Register 1 to the address of RAM pointed by the data of Register 2, SUBi stands for subbing with immediate data, BEQ R3-R0 stands for judging whether Register 3 and Register 0 are equivalent, J stands for jump directly, LD R3-R2 stands for reading the data of RAM which is pointed by the data of Register 2 and writing to Register 3, and WRT RL1-R3 stands for writing the data of Register 3 to the right output 1.

```
ADDi R2, R0, #0 // set R2 register value is zero
ADDi R3, R0, # 1920 // set R3 register value is 1920
RECEIVE: REC R1-RL3
ST R2-R1
ADDi R2, R2, #1
SUBi R3, R3, #1
BEQ R3-R0, #SENT
J #RECEIVE
SENT: ADDi R2, R0, #0
LD R3-R2
WRT RL1-R3
```

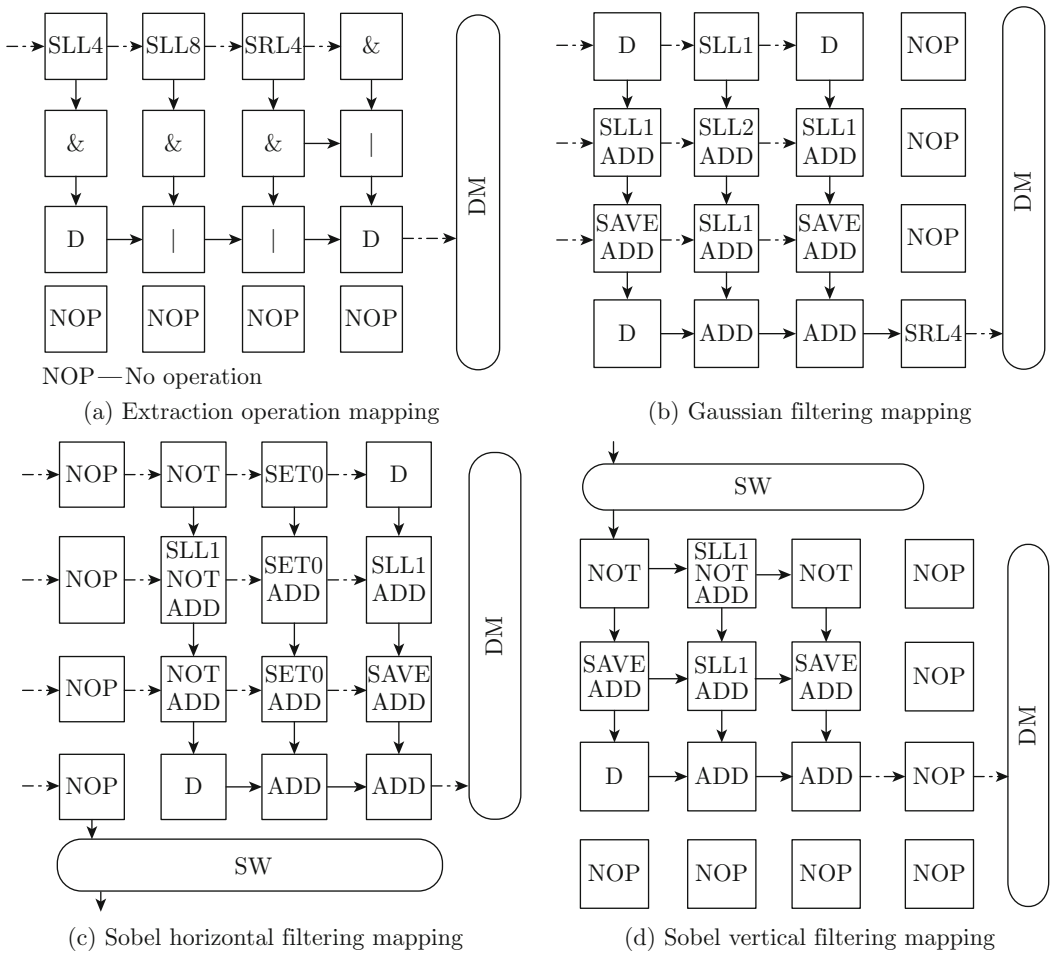
REC is mainly used to receive results from extraction operation. Because Gaussian filtering needs three-line data of image, there three-line data are saved firstly in the data distributed memory. The data distributed memory can be seen as FIFO. When three-line data are all saved, WRT is used to send data to Gaussian filtering cluster.

Data exchange between Gaussian node and Sobel node is similar to that between extraction node and



(a) Horizontal convolution (b) Vertical convolution

Fig. 13 Sobel filtering dataflow diagram



(a) Extraction operation mapping (b) Gaussian filtering mapping

(c) Sobel horizontal filtering mapping (d) Sobel vertical filtering mapping

Fig. 14 Application mapping diagram

Gaussian filter node.

4.2 CIC Filter

CIC filter[15] consists of an integral part and a comb part, and its system transfer function is given as

$$H(z)=\frac{1}{1-z^{-1}}(1-z^{-D}), \tag{4}$$

where D represents the number of cascade (here D is 3), and z represents the input signal.

In this implementation, three-level cascade and ten times extraction are employed. The dataflow graph of the integral part of CIC filter is shown in Fig. 15. Three vertices are used as accumulators to

compute[15]:

$$y(n) = y(n-1) + x(n). \tag{5}$$

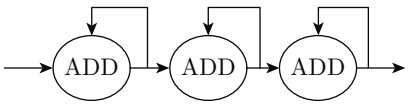


Fig. 15 The integral part of CIC filter

The dataflow graph of the comb part of CIC filter is shown in Fig. 16. Nine vertices are used as accumulators to compute[15]:

$$y(n) = x(n) + x(n-2). \tag{6}$$

According to those dataflow diagrams, we can easily map CIC filter onto DD-RCA. The mapping diagram is shown in Fig. 17, where dotted lines stand for data from or to distributed memory. The integral part is mapped onto left side, and the comb part is mapped onto right side.

Ten times extraction can be realized by configuring distributed memory. Configuration information is described as follows. REC R2-RL1 stands for receiving data from the left input 1 to Register 2, BEQ R1-R0 stands for judging whether Register 1 and Register 0 are equivalent, and WRT RL3-R2 stands for writing the data of Register 2 to the right output 3.

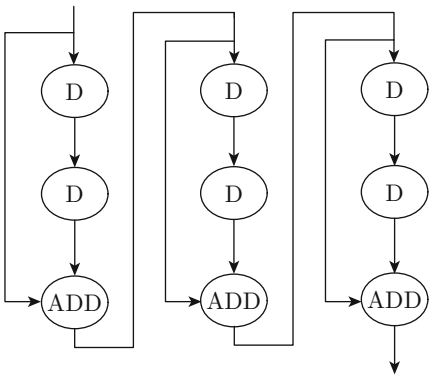


Fig. 16 The comb part of CIC filter

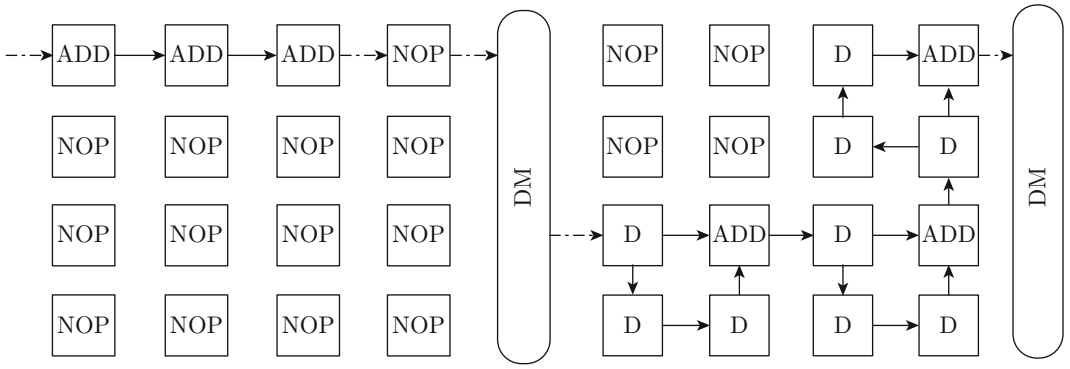


Fig. 17 CIC filter mapping on DD-RCA

```
ADDi R1, R0, #10
RECEIVE: REC R2-RL1
SUBi R1, R1, #1
BEQ R1-R0, #SENT
J #RECEIVE
SENT: WRT RL3- R2
```

5 Performance Study

The running results of computing vision process segment on DD-RCA are shown in Fig. 18. The image in Fig. 18(a) is original image. The image in Fig. 18(b) is blurred, compared with the image in Fig. 18(a). The images in Figs. 18(c) and 18(d) are the results from Sobel operator x-axis and y-axis filtering, respectively. Table 1 shows the performance of DD-RCA. In this table, "mapped node" indicates the number of cells used in a DD-RCA implementation for the particular component. For example, the extraction has ten DFG vertices, and these vertices are mapped onto twelve nodes in one cluster. "Mapping efficiency" is defined as the number of DFG vertices in a component divided by the number of mapped nodes. For the extraction, the mapping efficiency equals 83.3%. "Resource efficiency" is defined as the number of mapped nodes in a cluster divided by the total number of cells in the cluster. For the extraction, the resource efficiency is 75%. "Throughput" means the cycles for outputting one data.

From Table 1, we can see that the average throughput can reach one cycle per output data if one operation maps one cell. The average throughput cycles increase with increasing the number of operations mapped onto one reconfigurable cell. The number of DFG vertices is similar to the number of mapped nodes. So our architecture has a high mapping efficiency.

(a) Original image (b) Image through Gaussian filtering

(c) Image through Gaussian and Sobel x-axis filtering (d) Image through Gaussian and Sobel y-axis filtering

Fig. 18 The running results of computing vision process segment on DD-RCA

Table 1 Performance of DD-RCA

Component name	DFG vertex	Mapped node	Mapping efficiency/%	Resource efficiency/%	Throughput
Extraction	10	12	83.3	75	5
Gaussian filtering	14	13	108	81.25	3
Horizontal convolution of Sobel filtering	26	12	124	75	4
Vertical convolution of Sobel filtering	26	9	124	56.25	3
Integral part of CIC filtering	12	3	80	18.75	1
Comb part of CIC filtering	12	12	80	75	1

Based on the same input image with resolution of 512 pixel × 512 pixel, we compare the computing time of our architecture with that of GPU and CPU, as shown in Table 2. From this table we can see that DD-RCA has 11.3 speedup compared with CPU, but its computing time is longer than that of GPU because of low frequency of our architecture. However, in future our architecture will be realized on 90 nm or higher technology, and it will get 2—3 times acceleration. The computing time of our architecture will be comparable to GPU's.

Table 2 Computing time comparison of Sobel filtering on DD-RCA, GPU and CPU

Architecture	Computing time/ms
CPU[16]	102
GPU[16]	2.1
DD-RCA	9

The cells as well as the applications have been implemented on Xilinx FPGA xc6vlx550t chip. The circuit is able to run at a frequency over 100 MHz. The synthesis result is shown in Table 3. Basic unit includes one cluster, two reconfigurable switches and one distributed memory. One cluster consists of sixteen dataflow reconfigurable cells. All types include 4×4 basic units. From Table 3, we can see that our architecture consumes totally 192 222 registers and 28 774 look-up tables (LUTs).

The same circuitry has been synthesized using SMIC 130 nm COMS technology. The circuitry is able to run at a frequency over 200 MHz. The area is 12.84 mm^2.

Table 3 Synthesis result on Xilinx xc6vlx550t chip

Type	Number	
	slice/register	slice/LUT
Reconfigurable cell	664	965
Switch	272	257
Distributed memory	599	1 085
Basic unit	12 552	17 686
All	192 222	284 774

6 Conclusion

This paper presents a new RCA architecture. This is a dynamical reconfigurable storage cell array. This architecture is suitable for dataflow computation. It has some notable features: dynamical reconfigurability, static dataflow computation, specialized data exchange switch and data storage. Dynamic reconfiguration needs two operation modes: working mode and configuration mode. When a cluster is in configuration mode, it can receive and store configuration information. When a cluster is in working mode, it performs required computation. Some clusters can be in working mode while others are in configuration mode. The reconfigurable cells use the dual-rail protocol (REQ/ACK) to realize static dataflow computation and to free the computing cells from data exchange work load. The use of reconfigurable storage facilitates data exchange among clusters and makes the buffering of voluminous intermediate data easy.

References

[1] SMITH M C, PETERSON G D. Optimization of shared high-performance reconfigurable computing resources [J]. *ACM Transactions on Embedded Computing Systems*, 2012, **11**(2): 36.

[2] MIYAMORI T, OLUKOTUN K. A Quantitative analysis of reconfigurable coprocessors for multimedia applications [C]//*IEEE Symposium on FPGAs for Custom Computing Machines*. [s.l.]: IEEE, 1998: 2-11.

[3] SINGH H, LEE M H, LU G M, et al. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications [J]. *IEEE Transactions on Computers*, 2000, **49**(5): 465-481.

[4] VEREDAS F J, SCHEPPLER M, MOFFAT W, et al. Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes [C]//*International Conference on Field Programmable Logic and Applications*. [s.l.]: IEEE, 2005: 106-111.

[5] PACT XPP Technologies. XPP-III processor overview [EB/OL]. (2006-07-13). http://www.pactxpp.com.

[6] ZHU M, LIU L B, YIN S Y, et al. A reconfigurable multi-processor SoC for media applications [C]//*IEEE International Symposium on Circuits and Systems*. [s.l.]: IEEE, 2010: 2011-2014.

[7] LI T, XIAO L Z, HUANG H C, et al. PAAG: A polymorphic array architecture for graphics and image processing [C]//*International Symposium on Parallel Architectures, Algorithms and Programming*. [s.l.]: IEEE, 2012: 242-249.

[8] COMPTON K, HAUCK S. Reconfigurable computing: A survey of systems and software [J]. *ACM Computing Surveys*, 2002, **34**(2): 171-210.

[9] AMANO H. A survey on dynamically reconfigurable processors [J]. *IEICE Transactions on Communications*, 2006, **89** (12): 3179-3187.

[10] NAJJAR W A, LEE E A, GAO G R. Advances in the dataflow computational model [J]. *Parallel Computing*, 2000, **25**(13/14): 1907-1929.

[11] ROSENFELD J, FRIEDMAN E G. Design methodology for global resonant H-tree clock distribution networks [J]. *IEEE Transactions on Very Large Scale Integration Systems*, 2007, **15**(2): 135-148.

[12] Khronos Vision Working Group. The OpenVX specification [EB/OL]. (2014-10-07). https://www.khronos.org/registry/vx/specs/1.0.1/html/index.html.

[13] HOGENAUER E B. An economical class of digital filters for decimation and interpolation [J]. *IEEE Transactions on Acoustics Speech and Signal Processing*, 1981, **29**(2): 155-162.

[14] MONSON J, WIRTHLIN M, HUTCHINGS B L. Optimization techniques for a high level synthesis implementation of the Sobel filter [C]//*2013 International Conference on IEEE Reconfigurable Computing and FPGAs (ReConFig)*. [s.l.]: IEEE, 2013: 1-6.

[15] SHAN R, LI T, HAN J G. The buffered edge reconfigurable cell array and its applications [C]//*2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. [s.l.]: IEEE, 2013: 1023-1030.

[16] DORE A, LASRADO S. Performance analysis of Sobel edge filter on heterogeneous system using OPENCL [J]. *International Journal of Research in Engineering and Technology*, 2014, **3**(15): 53-57.