Pipelined FP Array for Stream-Driven Image Processor

Masahiro TABARA, Hiroki SHIBUTA, and Makoto IWATA
Graduate School of Engineering, Kochi University of Technology
Kami, Kochi, 782-8502 Japan

Abstract— *With the diverse spread of Internet of Things (IoT) technology, real time image recognition has been introduced into various application systems. In order to improve the recognition accuracy, higher frame rate, higher definition, and more advanced algorithm are progressing and its computational complexity thus tends to increase.*

In this paper, we propose the programing model and the architecture for stream-driven image processing (SDP-i). Stream-driven programming allows us to deal with a collective data structure as a stream in a program and it is represented as a directed graph so as to inherit the readability of the original dataflow model. In the proposed DDP architecture, we introduce a pair of ALU units and data memory into every stage of the FParray. After that, we conducted post-synthesis evaluation of the proposed DDP architecture by using 65 nm CMOS cell library. As a result, we verified that Hough transform can be executed in real time on dual DDP's.

Keywords: image processing, parallel processing, stream-flow programming, stream-driven processor

1. Introduction

With the diverse spread of Internet of Things (IoT) devices, real time image recognition has been introduced into various application systems, e.g., remote surveillance system, in-vehicle computer vision system, etc. They are indispensable for sustainable and safe society. In order to improve the recognition accuracy, higher frame rate, higher resolution, and more complex algorithms are required so that its computational complexity tends to increase more and more. This trend demands high-performance processors dedicated to the modern image processing in place of general-purpose microprocessors.

An image processing application is basically composed of several image processing components and most of those components are based on a calculation of a pixel or a set of pixels. In other words, it can be separated into two different operations; the macroscopic execution control of necessary image processing components and the microscopic pixel-stream-based processing of each component [1]. If we introduce two dedicated architectures for each of them, there is some possibility to achieve more high-performance as an image processor. Based on this consideration, this paper discusses a novel architecture of stream-driven image processor (SDP-i).

In order to design the SDP-i architecture, we first design a programming model suitable for the SDP-i. In this programming model called stream-flow programming (SFP) model, a program is hierarchically defined by several flow graphs and an arc (directed edge) in those graphs represents a data-dependency between nodes, on which some collective data types such as vector and matrix as well as scalar can be defined. Based on this SFP model, the SDP-i architecture is organized by an active function of generating and absorbing streams based on the data dependency and a passive function of processing the generated streams. The former is called a stream memory (SMem), the latter is realized by a processor core, data-driven processor (DDP) [2].

In order to supply necessary streams of the data to the DDP efficiently, the SMem must control macroscopic program structure to abstract a stream of data as a token in the SFP program. Therefore, all of the basic building blocks in the SMem operates at the level of the data stream. As for the DDP, a stream of data is decomposed into elemental data of the stream and they are operated at the finer level of grain. As a result of that, fine grain parallel processing capability inherent in the DDP is fully utilized. Based on those design consideration, this paper discusses architectural optimizations of the DDP and proposes its instruction set and micro-architecture from the viewpoint of spatial and temporal parallel processing, where parallel and pipelined execution of several elements can be realized on an array of function processing units called FParray. Finally, the evaluation results of the proposed DDP architecture are shown by designing 65nm CMOS circuit of the architecture and simulating it with some image processing programs.

2. Stream Flow Programming

While the pure dataflow programming model takes care of only the availability of primitive data called a token, the stream flow programming (SFP) described here is an extended programming paradigm of the dataflow model, which allows us to deal with a collective data structure as a stream in a program. The SFP program is also represented as a directed graph so as to inherit the readability of the original dataflow model [3]. In the SFP model, the following graphical notations of the program are defined; *port*, *arc*, and *node*. These notations are interpreted under the concept of stream flow model. The word *token* used in this model represents the existence of scalar data or a collective data when interpreting the SFP program. Furthermore, the SFP model defines a sequence of tokens as a stream.

- *Port*: Ports are defined for a module in an SFP program. The module normally has one or more input port(s) at its inflow point(s) and one or more output port(s) at its outflow point(s). The input port represents a kind of data source to produce a stream of tokens on its succeeded arc, and the output port does that of data sink to consume a stream of tokens flowing on its preceded arc. To define data on the input/ output port, data declaration must be described.
- *Arc*: An arc represents data dependency between operation nodes in the program. That is, a token is produced at the start point of the arc, and it is consumed at the end of the arc. If a stream of tokens is produced on the arc, each token flows on the arc in order. As for the program description, the arc is drawn between nodes, or a port and a node.
- *Node*: The node represents a function (primitive function, modularized function). The node is executable when there is a set of tokens associated with their corresponding input arcs. Even if all tokens belonging the stream are not available at the input arc, the function defined on the node may be evaluated for partial stream of tokens, i.e., so-called non-strict execution is supported in the SFP model.

Fig.1 shows an example of the SFP program. This program describes of matrix-vector multiplication with a matrix a[N][N] and a vector x[N]. The left-side graph MV-MULT in the figure represents an instantiation of a SFP module Ax with the matrix a and a vector x. The right-side graph in the figure defines the module Ax. In this graph, the port a generates a 2-D stream of data and the port x generates a 1-D stream of data N times iteratively. After that, the element-wise multiplication is executed at the first operation node $\times$, its resultant data is accumulated at the second operation node Σ, and finally the result of product-sum goes to the output port c. This operation is repeatedly executed N times. As a result, the result vector can be acquired. Each port is associated with the definition of its data structure described in the upper-left part of the SFP graph. As seen in this example, the SFP program can deal with a stream of streams and operate them consistently. This implies that every data structure which can be translated into a hierarchical stream is operable under the SFP model.

3. Stream-Driven Processor Architecture

The stream-driven processor (SDP) is based on an extension of the dynamic data-driven architecture from the viewpoint of the stream. Since the SDP executes the SFP program, SDP must generate and absorb a stream of data in non-strict manner at the SFP module level. On the other

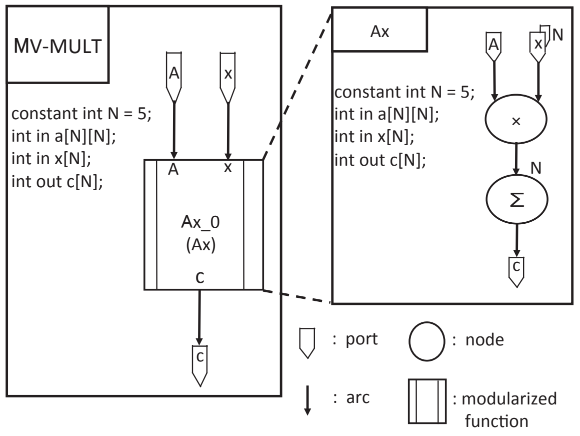

Fig. 1: An example SFP program (matrix-vector multiplication).

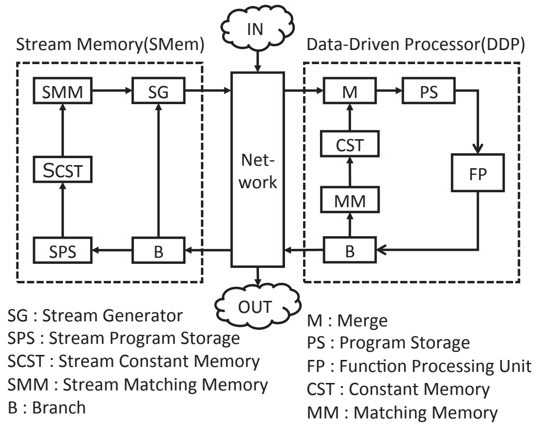

Fig. 2: Basic architecture of SDP.

hand, the SDP must support the data-driven execution at the primitive level such as an element data in a stream. In order to satisfy those functional requirements, the SDP is designed by combining both an active component and a passive component. The former called SMem is a stream memory realizing the functional module of the SFP port. The letter called DDP is a passive data driven processor realizing the function of processing elements of the stream. The SMem and the DDP communicate each other to transfer streams of data via the interconnection network. Fig.2 illustrates the basic architecture of the SDP where all data are dealt with in a packet format explained later. Furthermore, the SDP is fully implemented by the self-timed elastic pipeline (STP) circuit to explore the fine grain parallel processing described in the SFP program.

The SMem supplies a stream to the DDP in accordance with the SDP port description defined in the SFP program only when the stream becomes available for the operation node defined in the SDP program. The SMem handles the elements in stream at coarse grain to generate / absorb the streams efficiently. On the other hand, the DDP receives the supplied stream and executes primitive function nodes element-by-element according to the data-dependencies defined in the SFP program. Therefore, the fine grain parallel execution can be conducted on the DDP. After that, the resultant data stream produced by the DDP is returned to the SMem via interconnection network.

(a) Stream Memory

As shown in Fig.2, the SMem is structured with five functional units to conduct macroscopic program control of the SFP program at the stream level. Each unit of the SMem behaves as follows.

- Stream program storage (SPS)
  The SPS stores the SFP program. Each entry has the next module information which describes a couple of operand streams with their data structure definitions. The incoming packet representing a resultant stream fetches those information here and goes to the next SMM.
- Stream constant memory (SCST)
  The SCST stores constant data for the stream operand. The incoming packet representing an operand stream fetches a constant value here if its destination node needs a constant value for the operation. Otherwise, the packet passes through this unit.
- Stream matching memory (SMM)
  The SMM finds a pair of operand streams for the destination module described in the SFP program. The incoming packet representing an operand stream will be temporally stored here if the paired stream cannot found in the SMM. Furthermore, the incoming packet passes through this unit if the destination module needs a constant.
- Stream generator (SG)
  The SG stores all necessary stream data for the SFP program execution. In the branch unit, the resultant data stream is separated into a representative packet of the stream and stream element packets. The former packet is transferred to the SPS, and the latter packets are transferred to the stream memory of the SG. The incoming packet representing a stream is utilized to generate memory addresses and to read out necessary operand streams of data from the stream memory. In this case, the pair of operand data is dealt with as an element packet of the stream and its destination node identifier and index of the stream are attached to that packet. In this way, necessary number of element packets are generated in the SG and supplied to the DDP via the interconnection network.

(b) Basic Data-Driven Processor

The data-driven processor (DDP) plays a role of a passive component of the SDP. That is, supplied streams of operand-pair packets are processed in packet-by-packet (fine grain) according to the SFP program. The basic units composing the DDP are explained below.

- Merge unit (M)
  The merge unit arbitrates incoming packets from the SMem and circulating packets within the DDP to transfer those packets to the next PS. The arbitration policy employed here may be designed flexibly, but the basic one is the first-come-first-service.
- Program storage (PS)
  The PS stores the SFP program. Each entry has an operation code, a next destination node identifier, and some execution control flags. The packet arriving in the PS accesses its corresponding entry and fetches necessary information for executing the destination node.
- Function processing unit (FP)
  The FP is a kind of ALU. It accepts an operand-pair packet and executes an operation according to its operation code. Usually, one of the arithmetic and logic operation is executed and its result packet goes to the next unit.
- Branch unit (B)
  Based on the branch flag held in the incoming packet, the packet is output outside the DDP or inside it to execute the next operation.
- Matching memory (MM)
  The MM is responsible for the firing control of operation nodes. If there is no packet associated with the incoming packet, the packet is temporally stored in the MM and waits for the paired packet. If the incoming packet can be associated with the waiting packet, an operand-pair packet is created and output for execution. In the case of a unary operation or binary operation with a constant operand, the incoming packet passes through the MM without any packet manipulation.
- Constant memory (CST)
  The CST fetches a constant value from the constant memory, if the operation code held in the incoming packet indicates the binary operation with a constant operand. Otherwise, the incoming packet passes through the CST.

(c) Pipelined array of FPs

In order to execute image processing programs on the SDP-i efficiently, it is necessary to achieve better performance on both the SMem and the DDP. Furthermore, it is significant to optimize the potential bandwidth between them as well as to minimize the amount of necessary traffic within the interconnection network.

In order to increase the effective throughput in the DDP, an

Table 1: Extended packet format of SDP-i.

field	description	width
dest	Destination node	8 bit
color	Color	1 bit
age	Age	8 bit
dcore	Destination core	2 bit
osf	Operand stream flag	1 bit
sos	Start of stream	1 bit
eos	End of stream	1 bit
lc	Left carry flag	1 bit
lz	Left zero flag	1 bit
rc	Right carry flag	1 bit
rz	Right zero flag	1 bit
d0	1st data	32 bit
d1	2nd data	32 bit
d2	3rd data	32 bit
d3	4th data	32 bit

extended packet holding multiple data is introduced into our SDP-i design so that the spatial parallelism can be explored by making multiple FPs to work at the same time. Furthermore, the pipelined processing mechanism executing the compact module of data-driven sub-program is introduced in place of normal FP so that the temporal parallelism can be explored in the DDP. Those architectural optimization needs to redesign the normal DDP packet format and the instruction set dedicated to the image processing applications. Table 1 shows the extended packet format introduced in the SDP-i. In this packet format, the dest, color, age, dcore, osf, sos, and eos fields are necessary information to interface SMem and DDP. Since this packet can hold four 32bit words, it can fire two different binary operations at the same time. Thus, a pair of execution flags, carry flag and zero flag, are held in the packet.

In addition to this packet format, a couple of next operation codes and their related flags should be fetched at the PS of the DDP. Thus, the operand-pair packet format for dual FPs is defined as shown in Table 2. By using this packet format, dual operations of 32 bit operand-pair can be executed. Furthermore, a single operation of 64 bit operand-pair is also possible to be executed. As mentioned above, if multiple pairs of FPs are cascaded in pipelined structure, shuffle or swap of those operands is sometimes useful to execute small module of data-driven program directly on this pipelined FPs. Therefore, several shuffle instructions are introduced into the SDP-i. Moreover, the image processing often requires calculating complex mathematical functions such as trigonometric functions, square root, and so on. Since it is sometimes enough to calculate them roughly, the lookup table (LUT) stage for those functions may be introduced into the pipelined array of FPs. Data memory for small amount of temporal data (DMem) is also useful for most of image processing program. This also contributes to reduce the communication traffic within the interconnection network between the SMem and DDP. It is possible to combine the dual FPs, the LUT, the DMem flexibly in a pipelined array structure, which called FParray in this paper.

Table 2: Operand packet format for dual FPs.

field	description	width
dest	Destination node	8 bit
color	Color	1 bit
age	Age	8 bit
sos	Start of stream	1 bit
eos	End of stream	1 bit
lopc	Left operation code	8 bit
llr *	Left lr flag	1 bit
losf *	Left operand stream flag	1 bit
lwen *	Left write enable	1 bit
lc	Left carry flag	1 bit
lz	Left zero flag	1 bit
ropc *	Right operation code	8 bit
rlr *	Right lr flag	1 bit
rosf *	Right operand stream flag	1 bit
rwen *	Right write enable	1 bit
rc	Right carry flag	1 bit
rz	Right zero flag	1 bit
d0	1st data	32 bit
d1	2nd data	32 bit
d2	3rd data	32 bit
d3	4th data	32 bit

* not used in LUT and DMem.

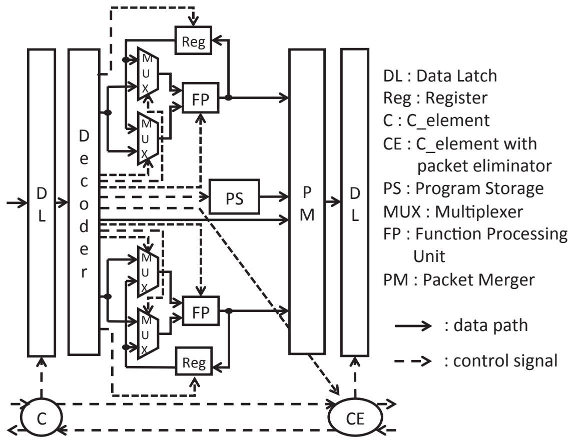

Fig. 3: Block diagram of dual FPs.

Some of the characteristic instructions in the FParray are listed in the appendix.

- C-element (C)
  The C-element is a basic data transfer control circuit of the STP stage. All C-elements deployed in the SDP-i autonomously behave. A couple of C-elements at the neighbor STP stages mutually communicate with each other, thereby controlling the transfer of data packets to the next stage. When transferring the data packet, the C-element generates the local clock signal to the data latch (DL).
- C-element with packet elimination (CE)
  In addition to the C-element, the CE can eliminate the data packet if the packet has a delete flag. When eliminating, the CE does not send the packet transfer

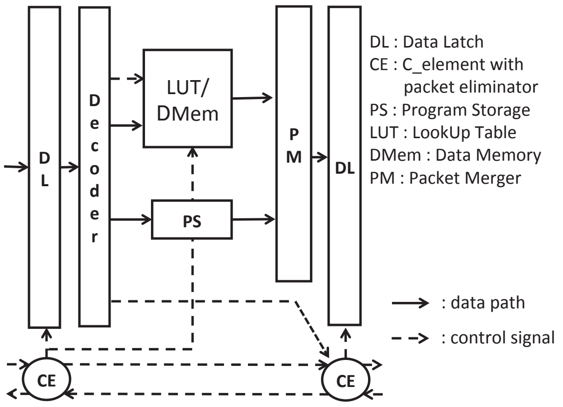

Fig. 4: Block diagram of LUT / DMem stage.

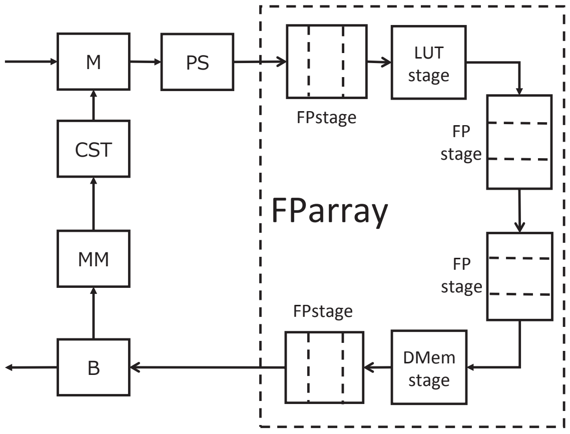

Fig. 5: Extended DDP with FParray.

request signal to the next C-element.

- Decoder
  The decoder asserts some control signals to MUX, FP, Reg, PS, and CE by decoding the operation code and flags held in the packet. Moreover, this module configures operand data necessary for MUX and PM.
- Function processing unit (FP)
  The function processing unit executes a primitive arithmetic operation along with the operation code. The operand data are supplied from the multiplexers. In the case of the product-sum operation, the result packet will be transferred to the accumulation register to be processed with the next data.
- Program storage (PS)
  The dual FPs possesses a small program memory that stores only the succeeded nodes. Thus, its function is same as the normal PS shown in figure 2.
- Packet merger (PM)
  Based on the control signal input from decoder, it reconstructs a pair of FP results into two pairs of operands.

The packet format of LUT / DMem stage is slightly different from the packet format of the FP. It is shown in table 2. In the table, * denotes that the fields are used in only LUT / DMem stage. The basic block diagram of them is shown in Fig.4. In the LUT stage, the lookup table is referred by the incoming packet and its result goes to the PM. In the DMem stage, the data memory is accessed by the direct address held in the incoming packet. When writing the data to the data memory, the data memory synchronously behaves with the local clock signal from the CE.

4. Evaluation

In order to evaluate basic performance of the extended DDP, the DDP equipped with FParray was designed by Verilog HDL and synthesized by Synopsys Design Compiler with 65 nm CMOS standard library. As shown in Fig.5, the FParray is composed of 6 pipeline stages: 4 FP stages, 1 LUT stage, and 1 DMem stage. In order to shorten the packet transfer time between the stages, it is necessary to minimize the critical path of every FParray stage. In this circuit design, the single FP stage is divided into three pipeline stages. The memory capacity in the LUT and DMem is dual 32bit words $\times$ 256 entries. In the SDP-i, the SMem, the interconnection network, and the extended DDP are constructed by 10, 4, and 19 STP stages respectively. If the packet dose not circulate within the DDP, the packet passes through only 17 stages because the MM and CST units are located between the branch and the merge units as shown in Fig. 5.

Table 3: Circuit cost of the extended DDP.

	FParray				DDP
	DMem	LUT	FP	total	total
std. cells	1.5K	1.3K	58.8K	247.1K	267.1K
area (mm^2)	0.067	0.060	0.185	0.753	0.866

The DDP circuit area excluding the memory is shown in Table 3. Since the SDP-i is implemented based on the self-timed pipeline, the basic performance parameters are defined as follows. That is, the packet transfer time at each pipeline stage is T_f, and the acknowledge time at the succeeded stage is T_r. Thus, the minimum periodic time between transferred packets at the stage can be defined by $T_f + T_r$. In the post-synthesis simulation, the critical path delay time of the FP stage is 11.4 ns. The longest T_f within the DDP is 15.5 ns in the matching memory MM. Thus, the achievable throughput of the DDP, operations per second, is restricted by the MM.

As a benchmark image processing program, we used the Hough transform detecting straight lines from an input image. The algorithm of Hough transform is as follows. The target image is assumed to be a binary-coded image including several straight lines.

- Hough transform: $\rho = x \cos\theta + y \sin\theta$
- Vote on frequent locations
- Hough inverse transform:

$y = -x/\tan\theta + \rho/\sin\theta$

Since the straight line detection of the w$times$h image is composed of the Hough transform, voting at high frequency positions, and Hough inverse transform. The execution time of each process, T_h, T_v, and Tih, can be estimated as follows.

$$T_h = (T_f + T_r) \times (wh + N_\theta N_v) \quad (1)$$

$$T_v = (T_f + T_r) \times N_\theta N_v \quad (2)$$

$$T_{ih} = (T_f + T_r) \times wN_l \quad (3)$$

where N_θ denotes the number of quantized angle θ, N_v denotes the number of valid pixels in the image, and N_l denotes the number of straight lines involved in the image.

If we introduce dual DDPs in the SDP-i, it is possible to execute the Hough transform and voting in parallel so that the execution time required for voting does can be hidden within the execution time of the Hough transform. As a result, the extraction of ten straight lines from UXGA (1600$\times$1200 pixels) image can be realized at 30 frame per second if we could employ ten DDPs.

5. Conclusion

An image processing application is basically composed of several image processing components and most of those components are based on a calculation of a pixel or a set of pixels. If we introduce two dedicated architectures for each of them, there is some possibility to achieve more high-performance as an image processor.

Based on this consideration, this paper discusses the programming model and the architecture of stream-driven image processor (SDP-i). The stream flow programming (SFP) is an extended programming paradigm of the dataflow model, which allows us to deal with a collective data structure as a stream in a program. The SFP program is also represented as a directed graph so as to inherit the readability of the original dataflow model. The SDP consists of the SMem actively generating and absorbing streams and DDP processing passive streams passively supplied. We examined application of this SDP for real-time image recognition and examined and designed SDP-i.

We designed the DDP equipped with FParray with parallel computing unit and memory function added. By introducing small data memory into the DDP, temporal intermediate results unnecessary for stream generation can be stored in the data memory in the DDP. Thus, the load of the SMem and the network part can be reduced. In addition, by parallelizing the operations of the FParray, it is possible to execute two kinds of operations at the same time. Based on the circuit design evaluation of the extended DDP, we confirmed that the simple feature extraction of 30 fps UXGA images can be realized on the SDP-i with ten DDP cores.

6. Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

This research work was supported in part by Japan Science and Technology Agency (JST). The circuit design work was supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] X. Lagorce, et al., “Asynchroous Event-Based Multikernel Algorithm for High-Speed Visual Featurs Tracking,” IEEE Trans. on Neural Net. and Learning Sys. Vol.26, No.8, pp.1710-1720, 2015.
[2] H. Terada, S. Miyata, and M. Iwata, “DDMP’s: Self-Timed Super-Pipelined Data-Driven Multimedia Processors,” Proceedings of the IEEE, Vol.87, No.2, pp.282-296, Feb. 1999.
[3] M. Iwata, H. Shima, and S. Sannomiya, “Self-Timed Stream Processor for Surrounding Computing Environment,” in proceedings of PDPTA’07-The 2007 International Conference on Parallel and Distributed Processing Techniques and Applications, pp.655–660, Las Vegas, USA, June 2007.
[4] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, “Low-Powered Self-Timed Pipeline with Runtime Fine-Grain Power Supply,” Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, pp.472-478, July 2012.

Appendix

Table 4: Instructions of FParray.

instruction	opecode	behavior
copy-xy	0000_XY	copy data-x into data-y
copy-agex	0000_00X	copy age into data-x
swap01	0000_0100	swap d0 and d1
swap02	0000_0101	swap d0 and d2
swap03	0000_0110	swap d0 and d3
swap12	0000_0111	swap d1 and d2
swap13	0000_1000	swap d1 and d3
swap23	0000_1001	swap d2 and d3
luta1-xy	0010_XY	load lutd-x into data-y
luta2-xy	0011_XY	load lutd0 into data-x and lutd1 into data-y
lda1-xy	0010_XY	load dmd-x into data-y
lda2-xy	0011_XY	load dmd0 into data-x and dmd1 into data-y
sta1-xy	0110_XY	store data-x into dmd-y
sta2-xy	0111_XY	store d0 into dmd-x and d1 into dmd-y

* : $\forall x, y \in \{0, 1, 2, 3\}, \quad \forall X, Y \in \{00, 01, 10, 11\}$