Evaluation of Space Allocation Circuits

Shinya Kyusaka[1], Hayato Higuchi[1], Taichi Nagamoto[1],
Yuichiro Shibata[2], and Kiyoshi Oguri[2]

[1] Department of Electrical Engineering and Computer Science,
Graduate School of Science and Technology, Nagasaki University,
1–14 Bunkyo-machi, Nagasaki 852-8521, Japan
{qsaka, higuchi, taichi}@pca.cis.nagasaki-u.ac.jp
[2] Department of Electrical Engineering and Computer Science,
Nagasaki University,
1–14 Bunkyo-machi, Nagasaki 852-8521, Japan
{shibata, oguri}@cis.nagasaki-u.ac.jp

Abstract. This paper describes the design and evaluation of the PCA (Plastic Cell Architecture) cell, which implements a novel space allocation method. PCA is a dynamically reconfigurable architecture which exceeds the FPGA (Field Programmable Gate Array) in flexibility and generality. Circuit dynamically reconfiguration is achieved as administrators manage the heap areas. But, because objects operate and require new space in parallel, it is difficult to manage them collectively. So, we introduced the concept of pressure, which enables space allocation. As a simulation result, we found that this new method, which relies on pressure commands, could solve the problems of object management efficiently. We designed the PCA cell with space allocation capability. Consequently, the number of gates per PCA cell is 200, and the maximum delay time per cell is 3.55 ns. Moreover, the 3×3 PCA cell processing of six space-allocation commands consumes 306.3μW.

1 Introduction

Real operations of functions are performed by hardware. To perform arbitrary function, the hardware itself must have a variable part. However, if the hardware is composed of only variable parts, it cannot be configured or given any logic. Therefore, hardware is composed of dual components – a variable part and a fixed part.

Based on this consideration, we conducted a study on a novel architecture called PCA (Plastic Cell Architecture), which is different from conventional computers in its realization of the double dual-structure. In the structure of the PCA, SRAM-type FPGAs (Field Programmable Gate Arrays) are scattered over fine-grain networks spanning all of the chip. By making the best use of this structure, it can directly perform computing, *e.g.*, memory storage, processing and information transfer in parallel. In addition, the structure and scale of the configured circuits can be adapted to the behavior of applications by transferring the configuration information of the FPGAs. A unit of configuration of circuits is called

an object in the PCA. Circuits are configured separately for each object and are added into the system on-the-fly. The objects in the system that become unused are stopped and detached. To make adding and removing objects on-the-fly easy, PCA has taken the approach of using asynchronous circuits, which have been widely recognized as a means of designing low power consumption circuits and alleviating the performance degradation caused by its global clock skew.

The most difficult problem has been space allocation in the PCA. Because objects move in parallel, it is hard to manage them collectively. Thus, the *pressure command* was proposed as a new space allocation method to solve the abovementioned problem. First, we created a simulator to evaluate the pressure concept, and then we developed three pressure command sets. Next, we decided to implement one command set and evaluate its performance. This evaluation is presented here.

The composition of this paper is as follows. In Section 2, a description of PCAs along with related work is presented as the background of this study. In Section 3, details of the mechanism of the pressure command are described. In Section 4, the pressure commands that we designed are described. In Section 5, the evaluation is described. The paper ends with a short summary in Section 6.

2 Background

2.1 Plastic Cell Architecture

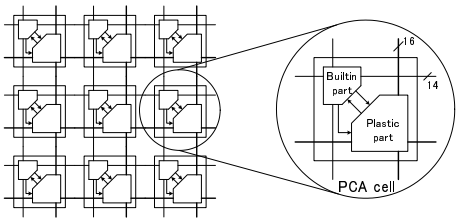


Fig. 1. Plastic Cell Architecture

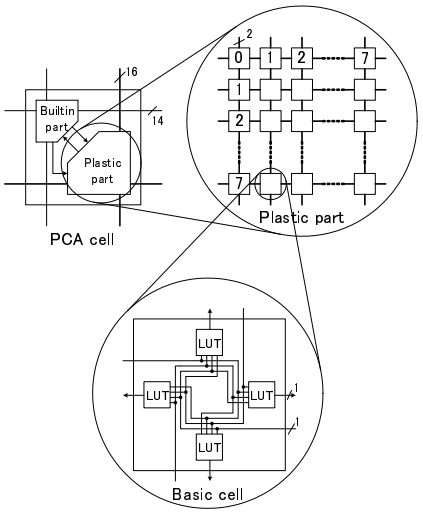


Fig. 2. Structure of plastic part

PCA (Plastic Cell Architecture) [1]–[3] is a dynamically reconfigurable architecture enhanced by the flexibility and the generality of the FPGA. PCA-1,

developed by NTT (Nippon Telegraph and Telephone Corporation), was the first LSI to realize this concept. This LSI is an array of PCA cells, each of which is composed of a plastic part and a built-in part (Fig. 1). The PCA cells are expected to realize the high integration of the LSI because of its homogeneous structure. Each PCA cell is connected with the adjoining cells in four directions. The plastic part works as both reconfigurable logic and local memory.

When it operates as a circuit, a circuit of arbitrary size is composed by connecting the plastic parts. When it operates as memory, the plastic part in the PCA cell is used with the unit. On the other hand, the built-in part connects the circuit and the memory in the plastic part and performs the data communication. The plastic part is composed of 8×8 two-dimensional basic cell mesh (Fig. 2). The basic cell is connected to four adjoining basic cells with 1-bit input and 1-bit output. Moreover, it has a 64-bit memory unit because it is composed of four 16-bit LUTs (look-up tables). This structure of the plastic cell that is composed of LUTs is called the Sea of LUTs.

The bit-serial PCA is an advanced PCA that adopts the asynchronous circuit system to a bit-serial data path. Asynchronous designs are clockless, so clock skews are avoided and an object can be designed with completely independent timing constraints. PCA operates by objects and messages, and the circuit structure based on local handshaking is also suited for miniaturization because it dispenses with long lines. A processing function is realized by state machines and shift registers in Bit Serial PCA. This coarse-grained composition can acquire efficiency higher than the Sea of LUTs.

2.2 Related Work

PCA-2 is being developed by NTT [4]. PCA-1 uses a $0.35-\mu$m process, but PCA-2 uses a $0.14-\mu$m process. The number of basic cells of the plastic parts are enhanced to 256, so the number of LUTs in a PCA cell are 1024. The width of the passing of data between built-in parts is enhanced to 9 bits. And the circuit composition for throughputs, such as making the detailed pipelines between modules, is changed. With such improvements, the performance has become better in the degree of integration and operation speed. PCA-Chip2, an experimental device of the PCA that uses a synchronous circuit, was developed at Kyoto University. The granularity that minimizes the amount of resources for logical synthesis was found by the method of synthesizing the benchmark circuit of MCNC to the various granularities [5]–[7].

The DRP (Dynamically Reconfigurable Processor) of NEC Electronics Corp. is a coarse-grained multi-context device [8]. The prototype chip DRP-1 was developed as the first LSI to use DRP architecture. DRP-1 has DRP cores, memory controllers, PCI controllers, and multipliers. The processing element (PE) of the DRP that is the reconfigurable element is the 8-bit arithmetic and logic unit/data management unit (ALU/DMU). The Digital Application Processor with Distributed Network Architecture (DAP/DNA), which IPFlex announced in 2002 [9], consists of parallel execution engine DNA matrices which are mounted for the exclusive use of the 32-bit RISC and 144 data processing

elements. The DAP RISC core controls the processor's dynamic reconfiguration, while portions of an application that require high-speed data processing are handled by the PE Matrix, which provides flexible parallel and pipelined operation. A DNA configuration memory consists of four banks. The Xpp of PACT XPP Technologies, Inc [10] consists of 128 32-bit processing array elements (PAEs). The PAEs and corresponding I/Os are segmented into two processing array cluster (PAC) blocks. A supervising confirmation manager (SCM) governs the configuration handling to local configuration managers, which further provide the necessary connectivity and processing for an algorithm.

3 Pressure Command

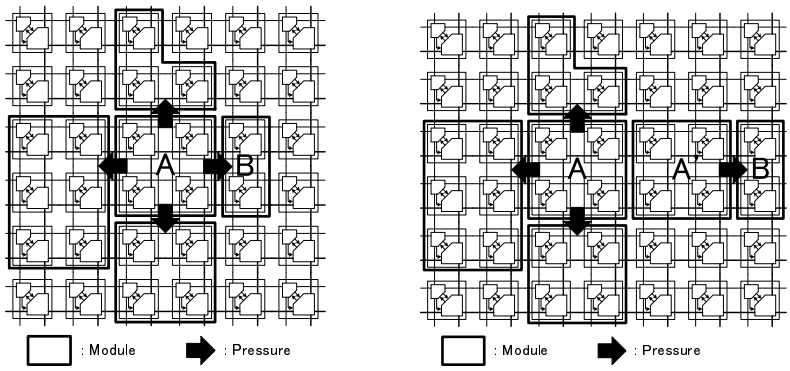
3.1 Space Allocation

The domain administrator of a conventional PCA manages space allocation. It receives the demand of a new domain from some objects, and pays out some new domains. But it has the following problems.

–An administrator has to always know all objects and vacant areas, because these consume unnecessary electric power. This ruins the advantage of an asynchronous circuit.

–It is a bottleneck of processing that a lot of requests concentrate on the administrator. This disrupts the essential parallel processing of the PCA.

–The method of reducing overhead by dividing the administrator has been proposed, but this method is thought to cause various problems because communication between the administrators is difficult.



(a) module A needs a new module

(b) module A makes a new module A'

Fig. 3. Pressure command

To solve the abovementioned problem, the *proliferation protocol* with the *pressure command* was proposed as a new method of space allocation. The concept is shown in Fig. 3. The communication among modules is performed through the built-in parts. Space allocation is also performed through the built-in parts with the proliferation protocol. If the module **A** needs the new module **A'**, it sends the pressure command to the surrounding modules, as shown in Fig. 3 (a). Then, the pressure command spreads to each module. The surrounding modules which accepted this command move to a vacant place. In Fig. 3 (b), the module **B** moved and the module **A'** is made in the vacated space. In this method, space allocation can be performed without an administrator.

3.2 Basic Proliferation Protocol

Although some methods of realizing this proliferation protocol are considered, the following is considered the basic protocol. In order to simplify problems, we decided that routing lines between objects are not considered and an object consists of one module (= one PCA cell). The basic proliferation protocol operates with five states and eight Pressure commands. The state of "EMPTY" means a module is not a meaningful module. The state of "ISMOD" means a meaningful module. The "DTSRC" module sends its modular data. The "TARGET" is a proliferation module. This module needs a new domain. The "DTDST" module receives modular data.

Each state of a module reacts reasonably to the received command, as shown in Fig. 4. If a module receives a command (**input-cmd**), the module sends a **command** in some **direction**s. *ST* means straight. *RETURN* means the module sends a command to itself. A module may send out two or more commands (shaded areas). And then, its state changes to *next-state*. The term **lock** has three conditions. *unlock* means that the module can receive commands from all directions. *BACK* means that the module can receive from only the one direction which received the command immediately before. Also, lock conditions may *remain*. How these commands are used is explained in the following section.

3.3 Example of Proliferation Protocol

As an example, the situation that displays the module (from module-1 to module-25) in the state 'ISMOD' to the procession of 5x5 is shown in Fig. 5. The procession outside closes with the module of state 'EMPTY'.

To attempt module-13's module cloning, the command START is given. When a module accepts the command START, the module's state changes to 'TARGET' and sends itself the command INCRZ, as in Fig. 5 (a). Sending the command INCRZ is continued until the proliferation operation is finished. If a module accepts this INCRZ command, it sends the pressure command to neighbor modules (Fig. 5 (b)). The module which receives the command PRESS2 (►) sends out the same command in the same direction. However, in addition to the same direction, the module which received the command PRESS1 (►) sends out a command PRESS2 to the right. The pressure spreads by transmitting this

0 EMPTY

input-cmd	next-state	command	direction	lock
ISEMPTY	0	FIX	BACK	unlock
PRESS1	4	ISEMPTY	BACK	BACK
PRESS2	4	ISEMPTY	BACK	BACK
ERASE	0	–	–	unlock
MAKE	0	FIX	BACK	unlock
INCRZ	0	–	–	unlock
FIX	0	–	–	unlock
START	1	–	–	unlock

(a) EMPTY

1 ISMOD

input-cmd	next-state	command	direction	lock
ISEMPTY	2	MAKE	BACK	BACK
PRESS1	1	PRESS1	ST	unlock
PRESS2	1	PRESS2 PRESS1	ST RIGHT	unlock
ERASE	1	–	–	unlock
MAKE	1	FIX	BACK	unlock
INCRZ	1	–	–	unlock
FIX	1	–	–	unlock
START	3	INCRZ	RETURN	unlock

(b) ISMOD

2 DTSRC

input-cmd	next-state	command	direction	lock
ISEMPTY	1	FIX	BACK	unlock
PRESS1	2	–	–	remain
PRESS2	2	–	–	remain
ERASE	0	–	–	unlock
MAKE	1	FIX	BACK	unlock
INCRZ	2	–	–	remain
FIX	1	–	–	unlock
START	2	–	–	unlock

(c) DTSRC

3 TARGET

input-cmd	next-state	command	direction	lock
ISEMPTY	3	MAKE	BACK	BACK
PRESS1	3	PRESS1	ST	remain
PRESS2	3	PRESS2	ST	remain
ERASE	1	–	–	unlock
MAKE	3	FIX	BACK	remain
INCRZ	3	INCRZ PRESS2	RETURN ALL	remain
FIX	3	INCRZ	RETURN	unlock
START	3	–	–	remain

(d) TARGET

4 DTDST

input-cmd	next-state	command	direction	lock
ISEMPTY	0	FIX	BACK	unlock
PRESS1	4	–	–	remain
PRESS2	4	–	–	remain
ERASE	0	–	–	unlock
MAKE	1	ERASE	BACK	unlock
INCRZ	4	–	–	remain
FIX	0	–	–	unlock
START	4	–	–	unlock

(e) DTDST

Fig. 4. State machine table

operation through the modules. When the module of state 'EMPTY' receives the pressure, this module sends out the command ISEMPTY in the opposite direction (Fig. 5 (c)) and its state changes to 'DTDST'. This means it has become the destination of modular data. The module which receives command the ISEMPTY changes its state to 'DTSRC' in order to move data to itself. At this time, that module ignores any pressure commands. The module of state 'DTSRC' sends out the MAKE command to an empty module, and passes the modular data. The module which receives the data with the command MAKE returns the command ERASE. Then, the original module changes to the state 'EMPTY', and is not a module. It is shown in Fig. 5 (d). As a result of the repetition of the movement of such a module, the module of state 'EMPTY' is made near to the module of state 'TARGET'. Even if several empty modules are in the neighborhood (Fig. 5 (e)), the 'TARGET' module chooses only one module as a candidate for a copy. Finally, the module of the target makes its own clone in an empty module (Fig. 5 (f)).

In this simple protocol, pressure acts on the meaningful modules and gravitation acts on the module which is not a module.

4 Design of PCA cell

This section describes the detailed inside specification and operation of the designed PCA cell which implements the pressure command.

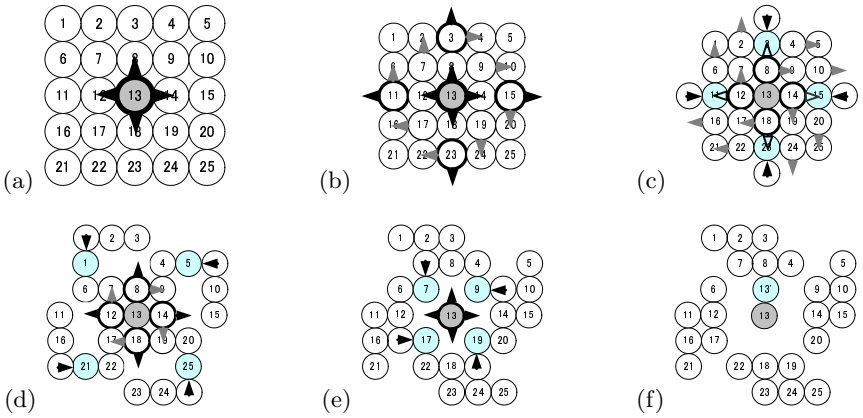


Fig. 5. Proliferation Protocol

4.1 Outline

We designed the PCA cell which implements the pressure command to evaluate the performance. Because this design focused on the justification and performance of the pressure command, the synchronous circuit system is adopted for design simplification.

4.2 Tools

SFL (Structured Function description Language), which is HDL (Hardware Description Language) developed at the NTT Research Institute for the logic composition tool PARTHENON (Parallel Architecture Refiner Theorized by NTT Original Concept), was used as HDL. SFL can make descriptions more elegantly than other languages can because it extracts the candidate for description to a single-layer synchronous circuit. 2SECONDS and PARTHENON were used as the simulator and logic composition tool for SFL, respectively. SFL2Verilog, which is one of the functions of PARTHENON, was used as a tool to change SFL into Verilog. and Design Compiler by Synopsis was used as the logic composition tool for Verilog.

4.3 Specification of PCA cell

The PCA cell is shown in Fig. 6. The concrete functions of each part are described in the following.

–Input and output

Because the number of pressure commands is eight, as mentioned in the previous section, the input and the output need only 3 bits to express them. Moreover, there is a 3-bit register for memorizing the input command at the input ports.

–"*lock*" and "*looked*"

Because one *State Machine* can receive only one command at once, there are two 4-bit registers called *lock* and *looked* inside. These are used to control the commands.

lock is used to carry out the command processing exclusively. Each bit corresponds as follows. Namely, bit 1 (MSB) is south, bit 2 is west, bit 3 is east and bit 4 is north. Data processing can be performed where the bit is 1. (*e.g.*, If the value of *lock* is 0b1000, it means that the state machine receives only the command from the north.) The initial value of *lock* is 0b1111.

looked is used in order to memorize the direction of a command entering and going out. Each bit corresponds as in *lock*. If the corresponding value is 1 when a command enters, it shows that the command enters from that direction. If the corresponding value is 1 when a command goes out, it shows that the command goes out to that direction. (*e.g.*, If the value of *looked* is 0b0010 when a command enters, it means that the command enters from the west.) The initial value of *looked* is 0b0000.

–*Judgment Part*

Input commands are sent to the portion called the *Judgment Part*. Here, the flag and *lock* of each direction investigate the commands before they enter the *Judgment Part*. If these bits are 1, the command from that direction can enter the *State Machine*. Moreover, as soon as the command goes into the *State Machine*, the bit of *looked* corresponding to the direction containing the command is set to 1, and the direction where the command was input is stored. If the following command doesn't enter from the direction that the command enters, the flag in that direction is adjusted to 0.

–*State Machine*

The *State Machine* receives input commands and determines the next command output and its direction. It can receive only one command at once. The *State Machine* has five states, and its processing is different from the command and the state described in the previous section.

5 Evaluation

We evaluated following items.

- Number of gates and maximum delay time of one PCA cell.
- Power consumption in an asynchronous circuit system of a 3×3 PCA cell (Fig. 7) processing six "START" commands.

We had to measure two power consumptions of the 3×3 PCA cell, i.e., processing six "START" commands and processing no commands, to determine the power consumption in the asynchronous circuit system. The power consumption of the 3×3 PCA cell processing six "START" commands consists of the power to process the actual commands and the power to make a clock signal. And the power consumption of the 3×3 PCA cell processing no commands contains only the power to make a clock signal. Therefore, the result, which subtracts

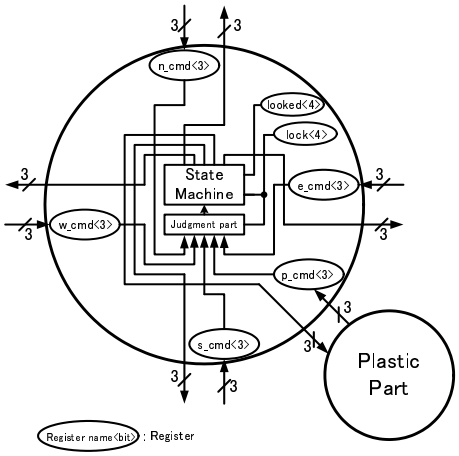


Fig. 6. PCA cell which implements the pressure command

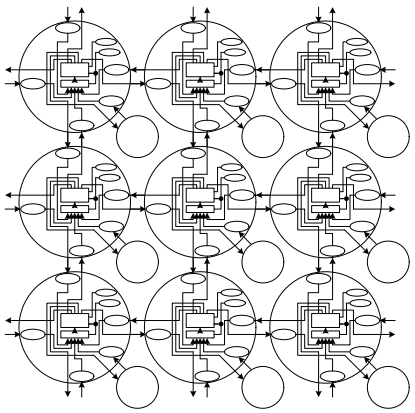


Fig. 7. 3×3 PCA cell

the latter from former, is the power consumption of the 3×3 PCA cell in an asynchronous circuit system which has no clock signals.

Number of gates of one PCA cell	200
Maximum delay time of one PCA cell	3.55 ns

Table 1. Evaluation of one PCA cell

The number of gates and the maximum delay time of one PCA cell is shown in Table 1. Since the number of gates of the built-in part in PCA-1 is about 2500, the ratio of the space allocation part is about 7%. The maximum delay time of the space allocation is lightly affected because its value is 25 ns in PCA-1.

Power consumption of 3×3 *PCA cell processing 6 commands*	16.0182 mW
Power consumption of 3×3 *PCA cell processing no commands*	15.7119 mW
Power consumption of 3×3 *PCA cell in an asynchronous circuit*	0.3063 mW (306.3 μW)

Table 2. Evaluation of 3×3 PCA cell

The power consumption of the 3×3 PCA cell processing the six "START" commands and processing nothing are shown Table 2. From these results, we could find the power consumption of the 3×3 PCA cell in an asynchronous circuit system. So, the power consumption of the 3×3 PCA cell per processing command is 51.0 μW.

6 Conclusion

This paper introduced command sets for the PCA. These command sets, based on the concept of pressure, were invented in order to realize a new space allocation method. After that, we implemented one command set in the PCA cell, and confirmed that it operated according to the protocol. The results of evaluating the performance of one PCA cell showed that the number of gates was 200, and the maximum delay time was 3.55 ns. Furthermore, the power consumption of the 3×3 PCA cell processing six commands in an asynchronous circuit system was 306.3 μW. Therefore, we found that the power consumption of the 3×3 PCA cell per processing command was 51.0μW. The ratio of the number of PCA cell gates (200) to the number of PCA gates of the built-in part (2500) showed that the space allocation part was only 7%. This fact indicates that it is possible to implement the space allocation part in the built-in part without large overhead.

Acknowledgment

The authors would like to express their sincere gratitude to the colleagues in their laboratory. The VLSI chip in this study was designed with PARTHENON, Synopsys Design Compiler, Cadence and Avant! CAD tools through the chip fabrication program of the VLSI Design and Education Center (VDEC) at the University of Tokyo in collaboration with the Rohm Corporation and Toppan Printing Corporation.

References

1. K. Oguri, "New general-purpose information processing architecture PCA1 by cloth line theory", bit, January 2000, Vol. 32, No. 1, pp. 27-35 Kyoritsu Syuppan (2000).
2. K. Oguri, "New general-purpose information processing architecture PCA1 by cloth line theory", bit, March 2000, Vol. 32, No. 3, pp. 54-62 Kyoritsu Syuppan (2000).
3. K. Oguri, "New general-purpose information processing architecture PCA1 by cloth line theory", bit, July 2000, Vol. 32, No. 7, pp. 51-59 Kyoritsu Syuppan (2000).
4. H. Ito, R. Konishi, H. Nakata, A.Nagoya, "Dynamically reconfigurable method LSI PCA-2", The 1st reconfigurable system society, pp. 119-126 (2003).
5. K. Soda, "Design of LUT array type PLD and method of burying logical function", Kyoto university graduate school graduation thesis, (2000).
6. Y. Soga, N. Sugimoto, T. Izumi, T. Ogata, Y. nakamura, "Examination of granularity of PCA-chip2 based on logical mounting efficiency", The 19th PARTHENON society", pp. 15-20 (1999).
7. N. Sugimoto, A. Tomita, H. TsuTsumi, K. Sakai, K. Soda, Y. Nakamura, "Design and making for trial purposes of LUT array type PLD", VDEC LSI design forum, September (2000).
8. M. Motomura, "A Dynamically Reconfigurable Processor Architecture", Microprocessor Forum,October (2002).
9. http://www.ipflex.com
10. http://www.pactcorp.com