Plastic Cell Architecture: A Dynamically Reconfigurable Hardware-based Computer

Hiroshi Nakada, Kiyoshi Oguri, Norbert Imlig, Minoru Inamori, Ryusuke Konishi, Hideyuki Ito, Kouichi Nagami, and Tsunemichi Shiozawa

NTT Optical Network Systems Laboratories
1-1 Hikarinooka Yokosuka-shi Kanagawa-ken 239-0847 Japan
{nakada, oguri, imlig, ina, ryusuke, hi, nagami, shiozawa}@exa.onlab.ntt.co.jp

Abstract. This paper describes a dynamically reconfigurable hardware-based computer called the Plastic Cell Architecture (PCA). PCA consists of dual-structured sea-of -cells that consist of a built-in part and a plastic part. The built-in part forms 'cellular automata' while the plastic part looks like an SRAM-based FPGA. We detail the design flow especially how to implement logic onto the processing element. The advantages of PCA, considering VLSI implementation and several application examples, are also described.

1 Introduction

During the past few years, 'system on silicon' which can realize a total information processing system on one chip has been widely discussed with the assumption of the future dense VLSI era. Most of the systems proposed, however, are just the integration of conventional elements onto one large silicon chip, where down sizing and performance improvement will be achieved as a natural consequence. Given the inherent limitations of this idea, an entire new approach to next generation computer systems is needed that will make the best use of 'system on silicon'.

In general, there are two approaches to execute process at higher speed on a programmable system; one is reducing the clock cycle (or shortening the critical paths) in the hardware, the other is to extract and utilize parallelism in the process. In other words, the former approach challenges the 'time domain' while the latter approach tackles the 'space domain'. The former approach has common in most of the processor-based systems that execute software described by the computer language. On the other hand, many combinations of FPGA-based systems and Hardware Description Language (HDL) have been proposed in recent years, and these takes the latter approach. Unfortunately, FPGAs have not yet become a viable alternative to general purpose processors.

We have proposed both types of reconfigurable systems for programmable telecommunication systems, namely, FPGA-type [1], and processor type [2]. Moreover, we also engaged in developing an HDL-based CAD system [3]. We revisited the loosely-coupled relationship between the HDL and reconfigurable devices again given our experience and the next generation in VLSI technology.

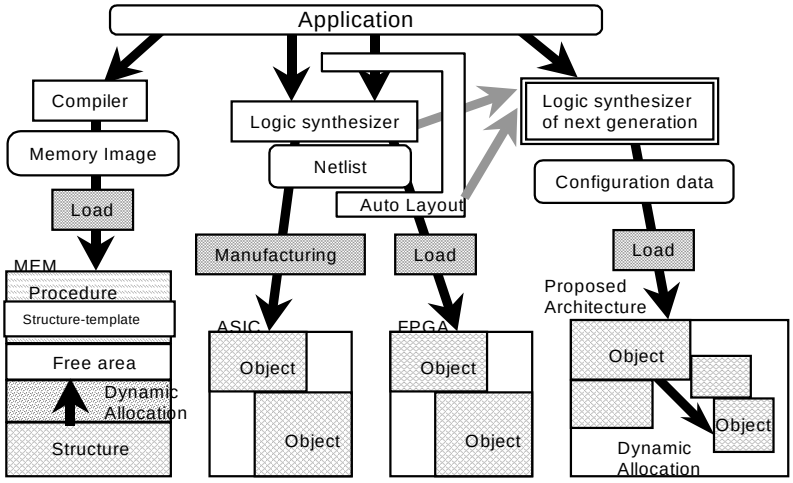

Fig. 1. The common and different points of implementation of application

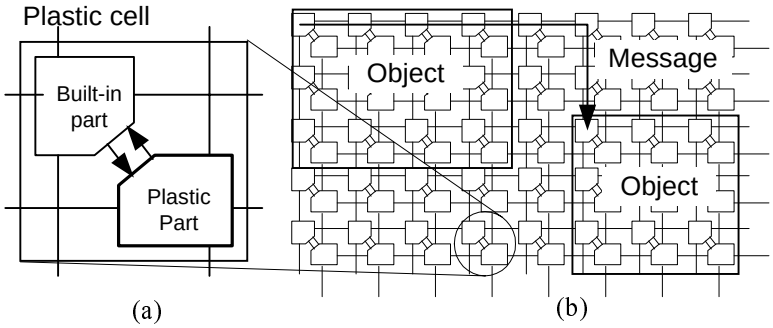

Fig. 2. (a) Plastic cell (b) Cell array with two hardware-objects communicating by message passing

Figure 1 compares implementing functionality in either software or hardware. While software is compiled into a memory image, hardware is synthesized into a static circuit image. Starting from this netlist representation a mask layout is generated for ASIC production in a fabrication or a configuration bit-stream for downloading into an FPGA. The major difference lies in the runtime behavior. Software supports dynamic memory allocation. On the other hand, dynamic reconfiguration is basically not supported by hardware. Since many algorithms operate on dynamically generated complex data structures, software was traditionally thought to be the only way to implement such functionality. Algorithms implemented as wired logic can also be realized by memory. Until the appearance of the FPGA, nobody thought about exploiting this dynamic nature of wired logic.

Our news approach to an original processing architecture has two aspects, namely, (i) HDL which provides the semantics of dynamic ‘generation’ and ‘deletion’ of the processing element, and (ii) hardware which realizes the above ‘generation’ and ‘deletion’ operation dynamically. We started this project from the basic hardware architecture.

This paper mainly describes the hardware mechanism and lower design process of the new reconfigurable computer, called the Plastic Cell Architecture (PCA). Section 2 describes the dual structure of PCA in detail. Section 3 mentions how to map logic to processing element of PCA. This involves a kind of VLSI layout problem. The advantage of PCA with considerations for VLSI implementation and applications are described in section 4. Finally, we conclude this paper by mentioning several related works and future study.

2 Architecture

2.1 Dual Structure

In order to allow dynamic runtime reconfiguration after the initial state has been set, a system must be composed of two distinct elements: a variable part as the place holder of the new configuration and a fixed part responsible for setting the former. In the example of a CPU-memory configuration, the memory corresponds to the variable part and the CPU to the fixed part. The same is true for FPGAs. Their configuration SRAM memory corresponds to the variable part while the attached CPU can be regarded as the fixed part.

Though our proposed architecture is also composed of two parts, the main differences from the above examples are as follows:

- Both parts are simple, and they are interconnected within a single LSI chip.
- The system consists of a huge mesh of above connected pairs (pairs of fixed and variable parts).

Figure 2.a shows the ‘plastic cell’ of our architecture. We call the variable part the ‘plastic part’ and the fixed part the ‘built-in part’. Both parts are connected. In order to exploit parallelism, the cells are arranged in an orthogonal array as indicated Fig.2.b. The corresponding parts of adjacent cells are also interconnected.

The functions of the built-in part are fixed and consist of plastic part reconfiguration, data I/O from/to plastic part, and mutual communication with the adjacent plastic parts. The array of built-in parts forms a network of ‘cellular automata’ [4]. A cellular automation is a processing model in which a collection of identical state machines exchange information with their neighbors by a simple set of rules. A plastic part is composed of several look-up table (LUT) memories, which can achieve various logic operations and a bypass function from one adjacent plastic part to another adjacent plastic part. This part can be reconfigured independently of other subsystems in the chip. The plastic part can also organize the memory object for data storage.

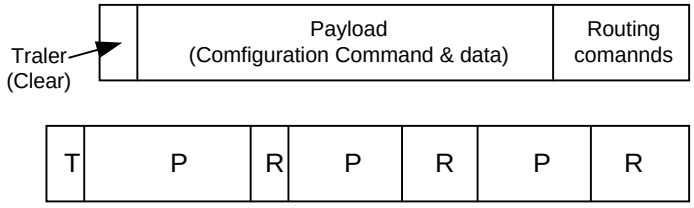

Fig. 3. Series of messages for inter-object communication (Two types)

2.2 Object and Communication

In a PCA, a processing module is called an 'object', its dynamic generation/deletion triggers the operation of another 'object'. For example, the output of an object '8-bit ALU' requires generation of an object '4-bit comparator', and so on. Objects are formed by several plastic parts, and generation, deletion, and data communication procedures are realized simply by message passing via built-in parts.

In order to dynamically allocate new objects the following steps are necessary:

1. Free area is searched and reserved
2. A new object is generated by injecting its behavior into the plastic part
3. The newly formed object is enabled for runtime operation
4. The communication path between built-in and plastic part is opened to allow message passing
5. Operation of the hardware object, communication with other objects
6. After receiving a release message, the area is freed for new allocation

Objects and generated objects have a mother-child relationship. Thus, only the mother object can send a release message and free the resources of an allocated child.

In order to enable proper configuration, the route of each message should be fixed. In fact, we employ exact routing rather than adaptive message routing. A series of messages is formed into a variable length packet as shown by Fig.3. Each packet is composed of a series of commands and data, which are categorized by three types of frames: routing command, payload, and trailer, which are processed by the built-in part. More detailed discussion for communication protocol was found in another

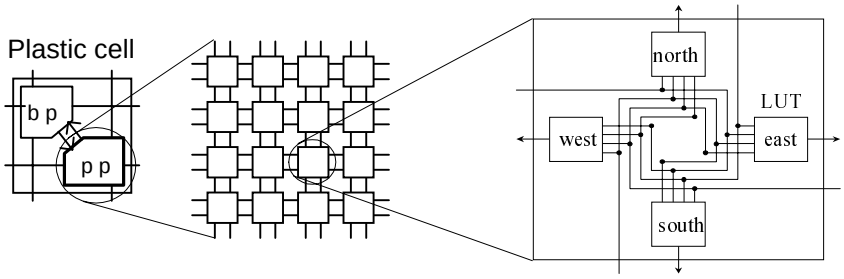

Fig. 4. Example of a plastic part architecture

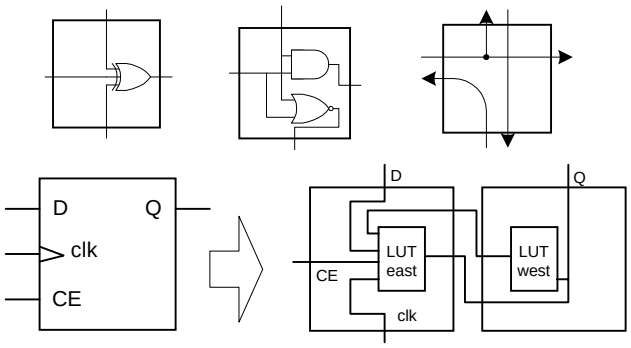

Fig. 5. Examples of logic circuit elements with plastic part

presentations [4],[5],[6].

2.3 Plastic Part and Asynchronous Circuit

Figure 4 shows an example of a plastic part architecture. Each plastic part consists of an array of 'basic cells'. A basic cell consists of four 4-input LUTs as shown in the figure. The most important feature is its regular structure. In general FPGAs, each cell has many components such as flip-flops, selectors, and transistor switches. On the other hand, a plastic part has only LUTs and wire. Only these components enable any binary logic operation for any direction of input/output signals, data branch, data transfer, and data storage. These examples are shown in Fig.5. This structure is called the 'sea of LUTs'.

One of the most remarkable points is that there is no dedicated flip-flop in the plastic part, that is, the flip-flop is constructed by combining and looping back of LUTs. Obviously it is not practical to use a global clock. Consequently, we regard all the logic circuit on the plastic part as asynchronous circuit, which has no clock signal. This is suitable for asynchronous pipeline architecture on for the built-in part.

3 Design Flow of Circuit on Plastic Part

In order to construct an object on PCA's plastic parts, the object is expanded into logic circuits described by netlists. At that time, a layout CAD technique is required for implementing netlists onto cells automatically and efficiently. As stated above, plastic part is a kind of SRAM type FPGA. In general, conventional FPGA layout system consists of following stages:

Stage 1: Technology mapping for technology independent logic circuits into LUTs.

Stage 2: Placement of each LUT.

Stage 3: Routing each wire between LUTs.

Usually, above each stage is optimized independently because general FPGA is so inhomogeneous that it is necessary to relocate one or more placed/routed modules in

order to insert a new cell/wire. In fact, if unrouted wires remain, it is necessary to back an upper stage and to re-optimize using different optimizing parameters.

On the other hand, PCA plastic part can be regarded as a 'relocatable PLA'. This feature is very useful to place complicated circuits compactly such as finite state machine. If PLA design flow is applicable for PCA layout, the output format of logic synthesis is only SOP (Sum of Products), and there need no special stage for placement and routing. Figure 6 shows an example of PCA layout image using PLA design.

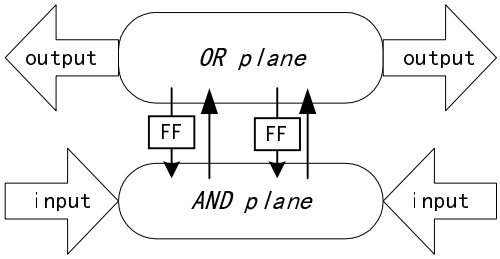

Fig. 6. Example of PCA layout image using PLA design scheme

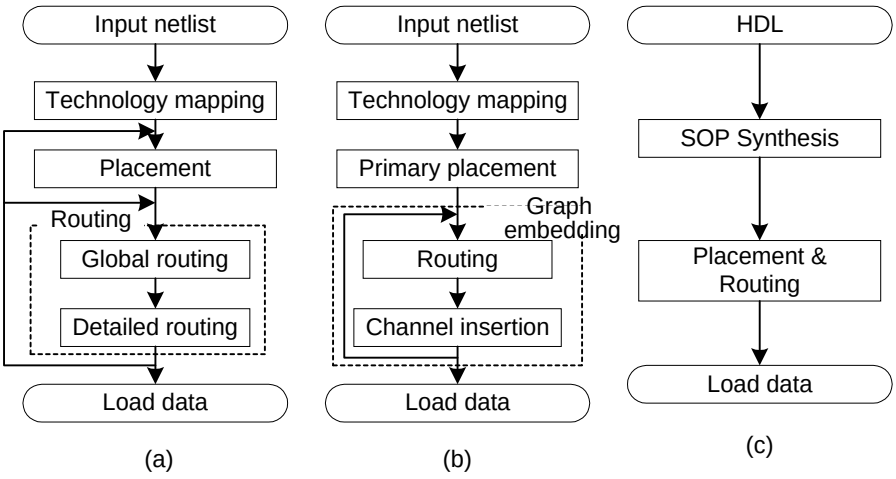

Fig. 7. (a) Conventional FPGA design flow (b) PCA plastic part layout design using conventional ASIC design technique (c) PCA plastic part layout design using PLA design scheme

Figure 7 compares PCA layout design flow to the usual FPGA design flow. As shown in Fig.7.b, conventional ASIC design technique is still useful for some cases, such as inter module communications or uneven distributed flip-flop circuits. Even in these cases, plastic part layout algorithm is simpler than usual FPGA case. For example, a classical LSI layout model (grid model) [8] for formulation and Dijkstra method [9] for routing algorithm are directly applicable for PCA plastic part layout.

4 Discussion

4.1 Advantage of PCA - VLSI Implementation -

In this section, we will discuss the advantages of PCA with regard to future VLSI implementation. As stated above, it is expected that memory fabrication technique can be applied to PCA because of its simple and regular structure. Figure 8 compares the number of transistors per chip of several types of VLSIs. In this figure, the Pentium II, fabricated using the newest 0.35micron technology, is composed of only 7.5 million transistors [10], while DRAM consists of 64 million transistors. On the other hand, the newest Xilinx FPGA contains 25 million transistors if 0.25micron technology is applied [10]. According to the SIA roadmap [11], the densest MPU using 0.25micron technology will place 11 million transistors in a chip. This shows that the regular memory structure significantly increases chip density. Of course, this comparison is not exactly fair because the object that should be compared is the actual logic circuit on the chip.

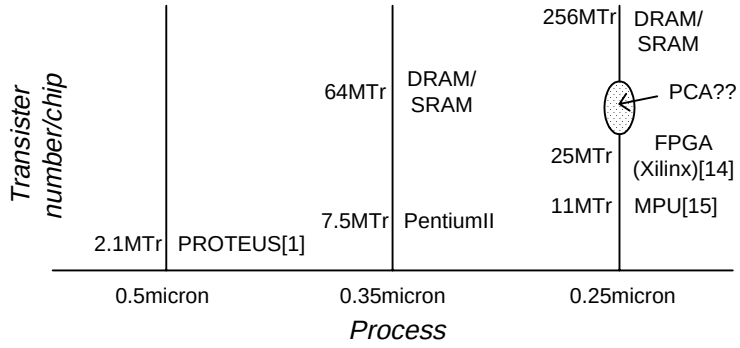

Fig. 8. Recent VLSI density trends [10],[11]

Table I Forecast of future VLSI chip density from SIA roadmap 1997 edition [11]

Year	1997	1999	2001	2003	2005	2009	2012
Process (nm)	250	180	150	130	100	70	50
DRAM (bit/chip)	267M	1.07G	1.7G	4.29G	17.2G	68.7G	275G
MPU (transistors/chip)	11M	21M	40M	76M	200M	520M	1.40B

Though above discussion itself is not so meaningful, consider the trend in VLSI technology. Table 1 is a forecast of future VLSI chip density [11]. It is noticeable that the density of memory will be quadruple every 3 years from now on, while MPU density will fair to double every 3 years. That is, while the difference in density between memory and MPU is about 24 times in 1997, the difference will increase to 56 times in 2003, and 200 times in 2012. Since even the regular FPGA can realize

higher density than MPU, the PCA, which has a pure memory structure in its plastic parts, will strongly benefit from this memory technology trend.

For this reason, it is expected that the scale of logic in a PCA will quickly catch up with that of an MPU given the trend in process technology. Let us consider 32-bit array multiplier for example. In general, a 32-bit array multiplier costs about 1000 logic gates because about 10 logic gates are required for a 1-bit full adder. This would require about 51 thousand transistors with ASIC realization if one gate costs four transistors. Suppose the same function is implemented on a PCA, it would costs about 2000 basic cells for a 32-bit array multiplier because a 1-bit full adder can be constructed with 2 basic cells including wiring. In total, about 1.3 million transistors are required for PCA implementation assuming 640 transistors per basic cell. Though PCA implementation wastes about 25 times more transistors than ASIC implementation, it differs by only 2.5 times in terms of area if the PCA VLSI is 10 times denser than the ASIC. Furthermore, if the difference of density widens over 25 times, PCA array multiplier would be smaller than the ASIC version. While this comparison is not directly useful because of built-in part area, the area difference of both realizations will converge in the near future.

4.2 Application - Asynchronous Multi-chip Systems -

In general, if one wants to construct a large system with FPGAs, multi-chip architecture is indispensable because of one FPGA has insufficient area. In this case, there are the following problems:

(P1) Pin neck problem occurs between chips,

(P2) Performance is degrade by chip interconnection,

(P3) Partitioning the circuit onto the chips is very difficult.

On the other hand, if the system is constructed using PCAs, the above problems will be resolved. The PCA has the following features:

(F1) Chip has pins only between built-in parts,

(F2) Communication between objects (or chips) is asynchronous and multiplexed,

(F3) Homogeneous and non-hierarchical structure,

Feature (F1) will solve problem (P1). Problem (P2) will be eliminated by setting high performance paths between chips. This is not so difficult given feature (F2) because communication between chips is the same as the communication between objects in a chip. Furthermore, architecture partitioning will not be a problem because feature (F3) is covers the chip level and also the multi-chip (or system) level.

Telecommunication systems are one example of an application that fully utilizes the above features. Most public telecommunication systems realize low layered high speed processing by synchronous circuits while IP packets or ATM cells are asynchronously multiplexed in the communication channel. This architecture demands complicated buffer structure. If we select the appropriate frame format for inter-object communication, PCA may become a key device for programmable telecommunication systems. The authors are now considering a telecommunication system model based on PCA.

5 Conclusion

A new dynamically reconfigurable hardware-based computer (PCA) was introduced in this paper. PCA, which is based on built-in and plastic parts, is a cross between an FPGA and cellular automata. A von Neumann architecture is obtained if only one cell part is extracted. The memory corresponds to the plastic part and the built-in part can be compared with a mini CPU engine.

Some runtime reconfigurable SRAM based FPGAs have recently entered the market. However, a CPU is required for configuration. A more decentralized configuration approach is proposed in the wormhole system [12] [13]. The difference between their approach and ours is the fine granularity of the PCA. A highly fault-tolerant cellular architecture that mimics nature and biological concepts was proposed in [14]. It is widely known that many algorithms can be speeded up if memory can realize useful logic functions as well as data storage. This is the case with the plastic part of our cell.

We are now simulating the architecture in order to balance performance, resource, and VLSI implementation issues. The enhancement of the HDL for ‘generation’ and ‘deletion’ operation is also being introduced.

References

1. Ohta, N., Nakada, H., Yamada, K., Tsutsui, A., Miyazaki T.: PROTEUS: Programmable Hardware for Telecommunication Systems. Proc. ICCD ’94 (1994) 178–183
2. Inamori, M., Ishii, K., Tsutsui, A., Shirakawa, K., Nakada, H.: A News Processor Architecture for Digital Signal Transport Systems. Proc. ICCD ’97 (1997) 157–162
3. Nakamura, Y., Ogu.ri, K., Nagoya, A.: Synthesis from Pure Behavioral Descriptions. In Camposano, R., Wolf, W. (eds.): High-Level VLSI Synthesis. Kluwer Academic Publishers (1991) 205–229
4. von Neumann, J.: The Theory of Self-Reproducing Automata. Univ. of Illinois Press (1966)
5. Nagami, K., Oguri, K., Shiozawa, T., Ito, H., Konishi, R.: Plastic Cell Architecture for general Purpose Reconfigurable Computing. Proc. FCCM ’98 (1998)
6. Ito, H., Oguri, K., Nagami, K., Konishi, R., Shiozawa, T.: Plastic Cell Architecture for General Purpose Reconfigurable Computing. Proc. RSP”98 (1998)
7. Oguri, K., Imlig, N., Ito, H., Nagami, K., Konishi, R., Shiozawa, T.: General Purpose Computer Architecture Based on Fully Programmable Logic. Proc. ICES ’98 (1998)
8. Ullman, J.D.: Computational Aspects of VLSI. Rockville, MD: Computer Science Press (1983)
9. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley Publishing Co., Inc. (1983)
10. Xilinx Inc.: XC4000XV Product Highlights. http://www.xilinx.com/products/xc4000xv.htm (1998)
11. Semiconductor Industry Association (SIA).: The National Technology Roadmap for Semiconductors, 1997 Edition. (1997)
12. Bittner, R., Athanas, P., Musgrove, M.: Colt: An Experiment in Wormhole Run-time Reconfiguration. Proc. SPIE Photonics East `96 (1996)
13. Luk,, W., Shirazi, N., Cheung, P.Y.K.: Compilation Tools for Run-Time Reconfigurable Designs. Proc. FCCM’97 (1997) 56–65
14. Nussbaum, P., Marchal, P., Piguet, Ch.: Functional Organisms Growing on Silicon. Proc. ICES’96 (1996) 139–151