Dynamically Reconfigurable Gate Array Based on Fine-Grained Switch Elements and Its CAD Environment

Masanori Hariyama, Waidyasooriya Hasitha Muthumala and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi,980-8579, Japan
Email: {hariyama@, hasitha@kameyama., kameyama@}ecei.tohoku.ac.jp

***Abstract*— Dynamically programmable gate arrays (DPGAs) promise lower-cost implementations than conventional FPGAs since they efficiently reuse limited hardware resources in time. One important issue on DPGAs is the large amount of configuration memory, which leads to area-inefficient implementation and large static power dissipation. This paper presents novel architecture of a switch block to overcome the required capacity of configuration memory. Our main idea is to exploit redundancy between different contexts by using a fine-grained switch element. The proposed MC-FPGA is designed in a 0.18μm CMOS technology. Its maximum clock frequency and the context switching frequency are measured to be 310MHz and 272MHz, respectively. The area of the proposed MC-FPGA is reduced to 45% of a typical MC-FPGA under a constraint of 8 contexts.**

I. Introduction

A dynamically reconfigurable gate array (DPGA) can be sequentially configured as different processors in real time, and efficiently reuses the limited hardware resources in time. One of the typical DPGA architectures is multi-context one. Multi-context FPGAs (MC-FPGAs) have multiple memory bits per configuration bit forming configuration planes for fast switching between contexts. However, the additional memory planes cause significant overhead in area and power consumption[1],[2]. Especially, switch blocks require a much larger memory capacity than look-up tables.

Figure 1 shows the typical structure of an MC-FPGA. An MC-FPGA consists of cells with a programmable logic block and a programmable switch block. The programmable switch block has a crossbar structure using multi-context switches. Each of the multi-context switches has multiple memory bits for multi-contexts and the configuration bit is selected from the memory bits according to the context ID. In the conventional approach, each switch requires N bits to store N contexts.

Most previous works for DPGAs reduce the overhead using device-level solutions. That is, compact memory devices such as DRAM cells were used to store configuration data[1].

To reduce the overhead of configuration memory in MC-FPGAs, this paper proposes an architectural-level solution based on the fact that there is redundancy in configuration bits between contexts. To illustrate the redundancy and regularity, Table 1 shows an example of configuration data of the switch block shown in Table I. Each row denotes configuration

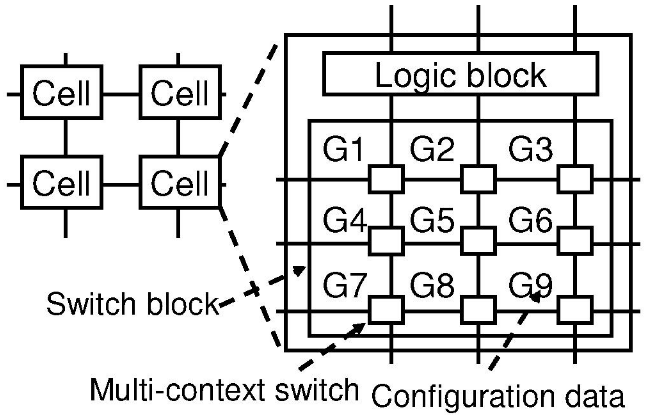


Fig. 1. Overall structure of an MC-FPGA

data of each switch. The configuration data G3 and G9 have redundancy in themselves. That is, there is no change in their configuration bits. It is said that less than 3% of configuration data are changed when contexts are switched[3]. There is another type of redundancy between configuration data of different switches. For example, G2 and G4 have the same configuration data. Moreover, there is a regularity in configuration data such as G2 and G4. The configuration data G2 and G4 can be represented by repeating bits in an order of (0,1).

To exploit the redundancy, we present fine-grained architecture for a switch block. The switch block consists of fine-grained switch elements with only a single multiplexer and two configuration bits. The switch elements are used in two ways: as programmable interconnections between logic blocks like conventional FPGAs, and as reconfigurable decoders that generate configuration bits from the context ID. By exploiting the redundancy in configuration data, the decoders become much smaller than the conventional multi-context switch.

II. Redundancy and Regularity in Configuration Data

For simplicity of explanation, an architecture with four contexts is considered although our approach is also applicable

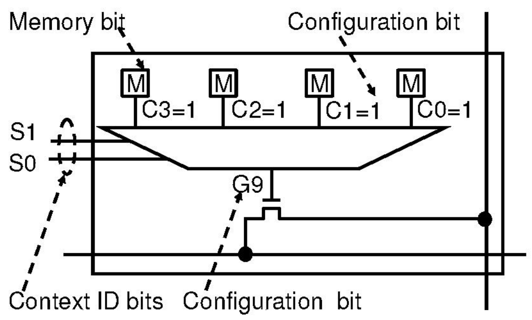


Fig. 2. Conventional multi-context switch (four contexts)

TABLE I
REDUNDANCY AND REGULARITY IN CONFIGURATION DATA

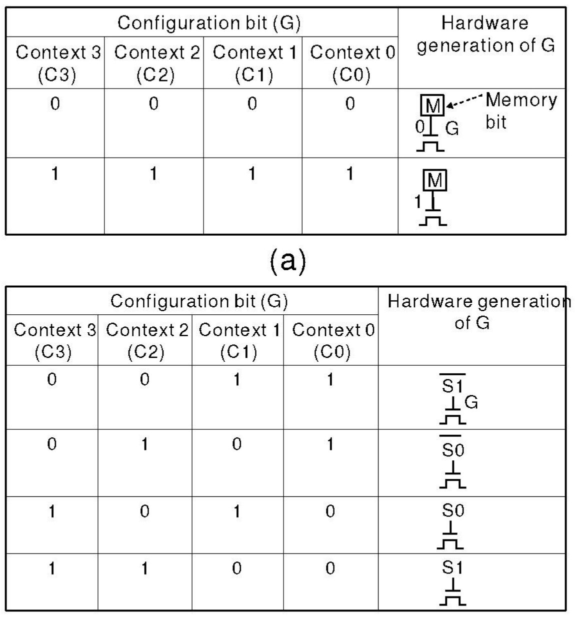
	Context 3 (C3)	Context 2 (C2)	Context 1 (C1)	Context 0 (C0)
G1	0	0	0	1
G2	1	0	1	0
G3	0	0	0	0
G4	1	0	1	0
G9	1	1	1	1

to the larger number of contexts. For four contexts, a 2-bit context ID (S1, S0) is sufficient. That is, context IDs (S1, S0)=(1, 1),(1,0), (0,1), and (0,0) represent context 3, 2, 1, and 0, respectively. This relation is summarized in Table I.

Figure 3 shows configuration-bit patterns with redundancy and regularity. The patterns in Fig. 3(a) have a redundancy. These patterns are independent from the context ID because the switch is always turned on or off. Therefore a single memory bit is sufficient to control the switch, while four memory bits are required for the conventional switch shown in Fig. 2. The patterns in Fig. 3(b) have a regularity, and they depend on a single context-ID bit. A switch using a single context-ID bit is smaller than the conventional switch that uses two context-ID bits. The other configuration-bit patterns depend on both of S1 and S0. Such bit patterns requires several multiplexers, and are slightly larger than the hardware shown in Fig. 3. However, such bit patterns does not frequently appear in a multi-context architecture since less than 3% of configuration data change when contexts are switched[3].

TABLE II
RELATIONS BETWEEN CONTEXTS AND CONTEXT ID BITS

	Context 3	Context2	Context 1	Context0
S1	1	1	0	0
S0	1	0	1	0



Configuration bit (G)				Hardware generation of G
Context 3 (C3)	Context 2 (C2)	Context 1 (C1)	Context 0 (C0)	
0	0	0	0	M Memory bit 0 G
1	1	1	1	M 1

(a)

Configuration bit (G)				Hardware generation of G
Context 3 (C3)	Context 2 (C2)	Context 1 (C1)	Context 0 (C0)	
0	0	1	1	$\overline{S1}$ G
0	1	0	1	$\overline{S0}$
1	0	1	0	S0
1	1	0	0	S1

(b)

Fig. 3. Configuration-bit patterns with redandancy.

III. SWITCH BLOCK ARCHITECTURE

Figure 4 shows the overall architecture of the proposed MC-FPGA. Basically, it has a cellular array structure like conventional MC-FPGAs shown in Fig. 1. The major difference between the conventional and the proposed ones is the switch block structure. The switch block of the proposed MC-FPGA is called Reconfigurable Context Memory(RCM). The RCM connects logic blocks(LBs). Moreover, double-length lines are used for high-speed data transfer. Figure 5 shows the structure of the RCM that consists of fine-grained switch elements (SEs), programmable switches (denoted by P) and input controllers (denoted by C). The programmable switch P connects a vertical track with a horizontal track. An input controller is used to invert its input. A SE consists of a pass-gate, a multiplexer and two memory bits (D1 and D0) as shown in Fig. 6. The SE is designed in such a way that the configuration-bit patterns with redundancy and regularity are implemented using a single SE. For example, in order to implement the second pattern in Fig. 3(a), D1 is set to 0 to select the constant input, and the constant input D0 is set to 0. In order to implement the fourth pattern in Fig. 3(b), D1 is set to 1 to select the variable input(U), and U is connected to S0.

The other complex configuration-bit patterns, which depend on both of S1 and S0, are implemented using several SEs. However, the area of them is much smaller than that of the redundant and regular patterns since such complex patterns don't frequently appear.

IV. EVALUATION

A test chip for the RCM-based MC-FPGA with 3×3 cellular array is designed in a 0.18μm CMOS design rule(Fig. 7). Its maximum clock frequency and the context switching

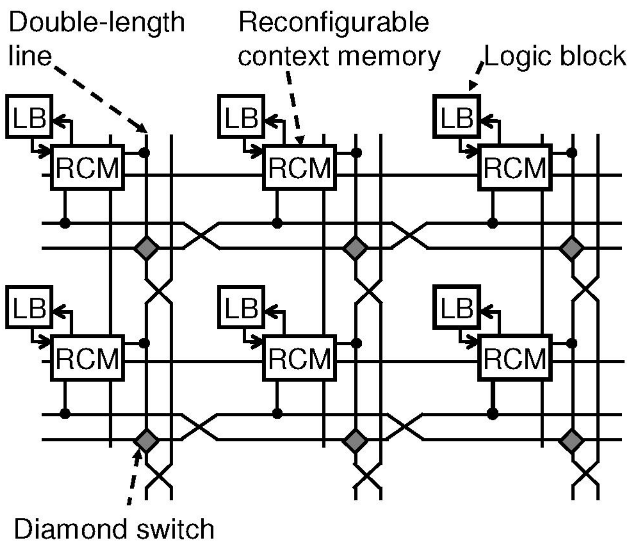


Fig. 4. Architecture of the proposed MC-FPGA

frequency are measured to be 310MHz and 272MHz, respectively. The MC-FPGA is compared with a typical one which uses switch blocks and logic blocks with fixed-size context memory for 8 contexts. We assumed that the percentage of changes in configuration data between contexts is 5%. Under a constraint of 8 contexts, an area of the proposed MC-FPGA is reduced to 45% in comparison with that of the typical MC-FPGA.

V. CAD Algorithm

Behavioral description of a design is given as a data-flow graph (DFG) with nodes and edges. A node represents an operation and an edge between two nodes represents the data dependency between the two corresponding nodes. Our design flow for the proposed MC-FPGA consists of 3 steps.

Step 1: Finding correspondence between the contexts.
Step 2: Combining DFGs.
Step 3: Placement and Routing.

The major differences between conventional CAD algorithms and ours are steps 1 and 2. Step 1 finds the sub-DFGs in different contexts that have the same structure, i.e. the same topology and mapped them into same cells to make simplified interconnections. For example, Fig. 8 shows 4 different DFGs assigned into 4 contexts and their common parts or the sub-DFGs. The map 1 in Fig. 9 is a random mapping of nodes into cell array. The map 2 in Fig. 9 assigned the corresponding sub-DFGs in the 4 contexts into same cells. As shown in Fig. 10, the map 2 gives a simplified interconnection than the map 1. As a result, the map 2 needs lesser interconnecting resources than the map 1 and that leads to reduce area and power consumption. According to step 1, the mapping algorithm maps the corresponding parts or the sub-DFGs into same area in the cell array of the MC-FPGA. However, this may cause some routing problems in the remaining nodes of the DFGs, because step 1 does not consider the topology of the rest of the

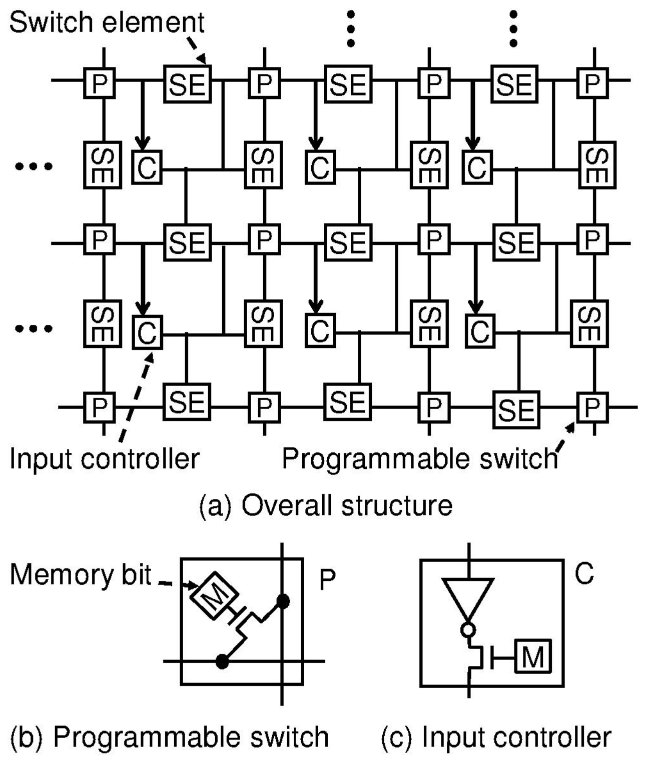
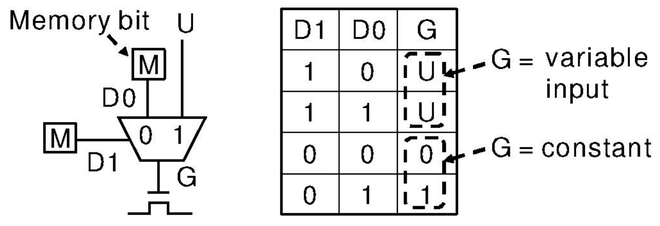


Fig. 5. Switch block using fine-grained switch elements.



D1	D0	G
1	0	U
1	1	U
0	0	0
0	1	1

Fig. 6. Fine-grained switch element

nodes. Therefore, we propose step 2 that makes a super-DFG by combining the given DFGs based on their correspondence. Fig. 11 is an example of a super-DFG that derived from the 4 DFGs in Fig. 8. We perform placement and routing algorithms based on this super-DFG. Let us show an experimental result for three different contexts: Smoothing with 111 nodes, Edge detection with 101 operations, and Template matching with 173 operations. The proposed mapping algorithm found sub DFGs with 95 nodes in the contexts. Therefore, as for smoothing and edge detection, more that 90% cells are shared in the different contexts.

VI. Conclusion

It is said that the static power due to leakage currents of SRAM cells occupies more than 30% in state-of-art technology smaller than 90nm process. Therefore, the proposed method is useful in the future process in terms of power consumption as well as the area (cost).

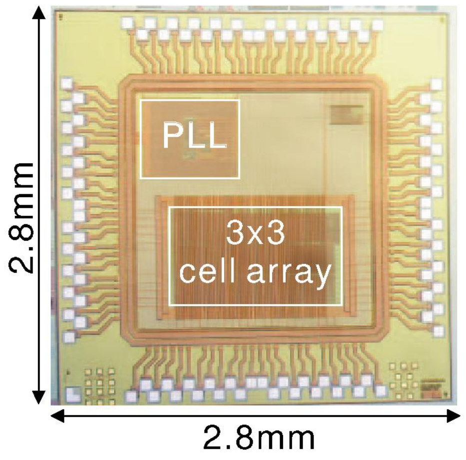


Fig. 7. Die microphotograph

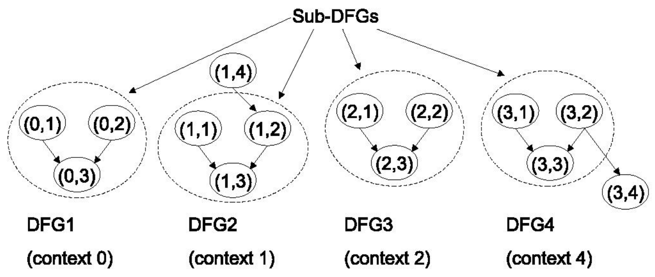


Fig. 8. DFGs in the 4 contexts

ACKNOWLEDGMENT

This work was supported in part by Industrial Technology Research Grant Program from New Energy and Industrial Technology Development Organization (NEDO) of Japan.

REFERENCES

[1] A. Dehon, "Dynamically Programmable Gate Arrays: A Step Toward Increased Computational Density", Proc. the Fourth Canadian Workshop on Field-Programmable Devices,pp. 47-54(1996).

[2] S. Trimberger, et al. "A Time-Multiplexed FPGA", Proc. of FCCM, pp. 22-28(1997)

[3] I. Kennedy, "Exploiting Redundancy To Speedup Reconfiguration of An FPGA", in Proc. FPL, pp. 262-171(2003)

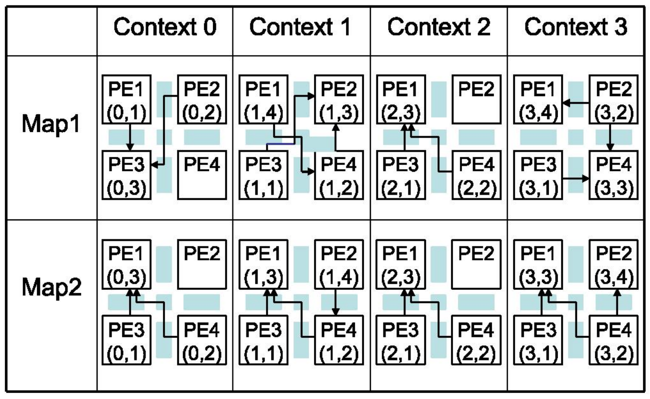


Fig. 9. Mapping example

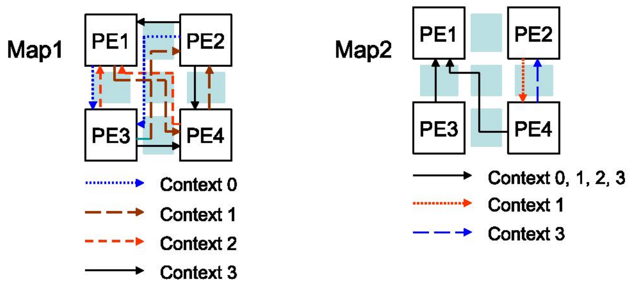


Fig. 10. Routing

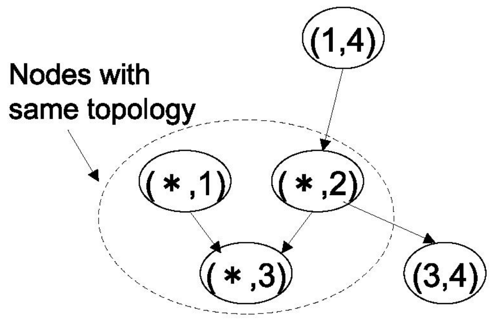


Fig. 11. Super-DFG