# POWER REDUCTION TECHNIQUES FOR DYNAMICALLY RECONFIGURABLE PROCESSOR ARRAYS

*T. Nishimura, K. Hirai, Y. Saito, T. Nakamura, Y. Hasegawa, S. Tsutsusmi, V. Tunbunheng,* and *H. Amano*

Department of Information & Computer Science, Keio University
3-14-1 Hiyoshi, Kohokuk-ku, Yokohama, 223-8522 Japan
email: muccra@am.ics.keio.ac.jp

## ABSTRACT

The power consumption of Dynamically Reconfigurable Processing Array (DRPA) is quantitatively analyzed by using a real chip layout and applications taking into account the reconfiguration power. Evaluation result shows that processing power for PEs is dominant and reconfiguration power is about 20.7% of the total dynamic power consumption.

Based on the above evaluation results, we proposed two dynamic power reduction techniques: functional unit-level operand isolation and selective context fetch. Evaluation results demonstrate that the functional unit-level operand isolation can reduce up to 20.8% of the dynamic power with only 2.2% area overhead. On the selective context fetch, the power reduction is limited by the increasing of the additional hardware.

## 1. INTRODUCTION

Coarse grained dynamically reconfigurable processor arrays (DRPAs) have been received an attention as a flexible and efficient off-loading engine for various types of System-on-Chips (SoCs). Some devices are commercially available [1, 2, 3, 4, 5, 6, 7], and some of them have been used in devices for consumer electronics [8].

In order to achieve better area- and power-efficiency compared with traditional field-programmable devices such as FPGAs, they incorporate the following properties; a simple coarse grained processor consisting of an ALU, a data manipulator, a register file and other functional modules is used as a primitive processing element (PE) of an array, and dynamic reconfiguration of an PE array which enables time-multiplexed execution is introduced.

Since the main target of DRPAs is streaming processing used in battery driven hand-held devices, power consumption is one of essential issues. Such devices have natural advantages for low power operation. First, by using efficient parallel processing with PE array, high performance can be achieved with relatively low clock frequency. Second, by using time multiplexing operation with dynamically reconfiguration, only required part of algorithm is performed on the relatively small amount of hardware. It saves the power consumption for clock distribution and leakage required by large hardware. On the other hand, DRPAs require extra power for dynamic reconfiguration especially in a multi-context style DRPA which changes its configuration frequently.

Although consuming power of particular DRPAs has been reported[1][9][10], the power used for dynamic reconfiguration, PEs and interconnection network has not been well analyzed. The effect of applying low power techniques proposed for traditional processors and FPGAs have not been well investigated.

Here, we analyze the power consumption of a DPRA with realistic layout and practical applications. Then, low power techniques are applied, and their effects are evaluated.

## 2. A TARGET ARCHITECTURE: MUCCRA-P

Here, a model architecture for power analysis, MuCCRA-P is introduced. MuCCRA-P is a small scale, multicontext DPRA designed for this power analysis. The architecture is almost the same as MuCCRA-2[11] which is now working on a real chip, but chip dependent design optimization is eliminated for analyzing a common design. The evaluation and modification can be done using the layout of the chip design.

### 2.1. PE ARRAY

As shown in Figure 1, MuCCRA-P has a $4 \times 4$ PE array and four distributed memories (MEMs) which have 32 bit $\times$ 256 entries on the bottom of array. For multi-media processing, the granularity of the whole architecture is 32 bits, that is, all functional units and channels treat 32-bit data except wires for 2-bit carry. Task Configuration Controller(TCC) and Context Switching Controller(CSC) are provided to manage task and context switching.

An island-style interconnection structure like traditional FPGAs is adopted for connecting PEs and MEMs. That is, each PE is surrounded by programmable routing wire segments. Connection blocks are provided between PEs and global routing channels for sending or receiving to or from PEs. On the intersection of vertical and horizontal channels, a Switching Element (SE) is placed. The SE is a set of simple programmable switches in which an entering link is connected to the other SEs. There are three channels for the global routing resources.

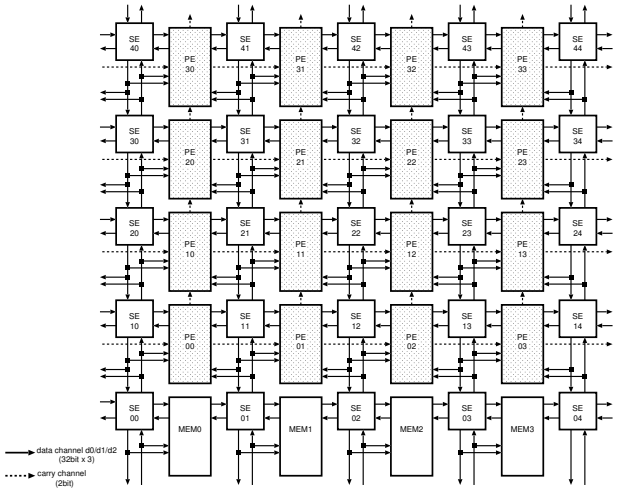


**Fig. 1**. PE Array Architecture

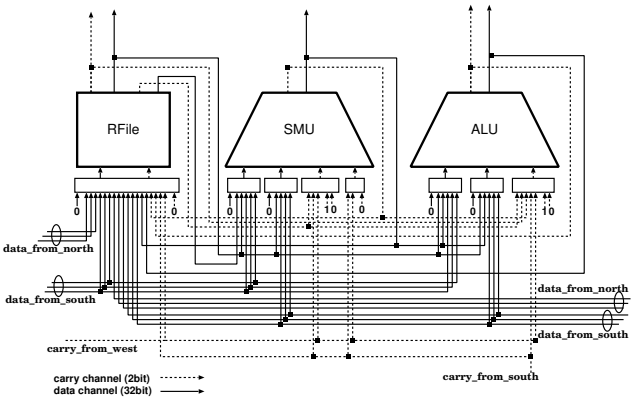


**Fig. 2**. PE Core Architecture

### 2.2. PE Structure of MuCCRA-P

Each PE is consisting of a programmable PE Core, connection blocks, and a context memory. In the PE Core as shown in Figure 2, like a lot of existing DRPA devices, a data manipulator called Shift & Mask Unit (SMU), an Arithmetic Logic Unit (ALU), and a register file (RFile) are provided. RFile has 34-bit (32 data bits + 2 carry bits) $\times$ 8 entries. A context memory which is 64 bits $\times$ 32 entries holds the configuration data that is distributed from the configuration memory at the beginning of the execution.

Each PE is connected with global routing wires via connection blocks. The connection blocks pick up the data from global routing wires, and distribute to all functional units of a PE Core. The operation of each functional unit and local intra-PE connection are defined by configuration data stored in the context memory.

### 2.3. Interconnection Network

The inter-PE connection network of the MuCCRA-P consists of Connection Blocks and SEs. Each PE can select and take data from vertical global routing resources (d0, d1, d2) in both sides of the PE via an internal input connection block called PICKIN. On the other hand, all outputs of ALU, SMU, and RFile of a PE can be transferred to horizontal links in any direction via the output connection block (PICKOUT).

Output data from a PICKOUT of a PE is transferred to PICKINs of other PEs through SEs. Each SE consists of four multiplexer-based programmable switches (SWs) and a context memory containing configuration data which specifies a destination of each SW. The SW transfers an entering data to desired output direction for each link (d0, d1, d2). Note that an input from the North is latched into a register in order to avoid combinatorial loops in the interconnection network, while inputs from other directions are unrestricted. The configuration data for a context of SE is 15-bit wide, and each context memory of SE can hold up to 32 contexts as similar to PE.

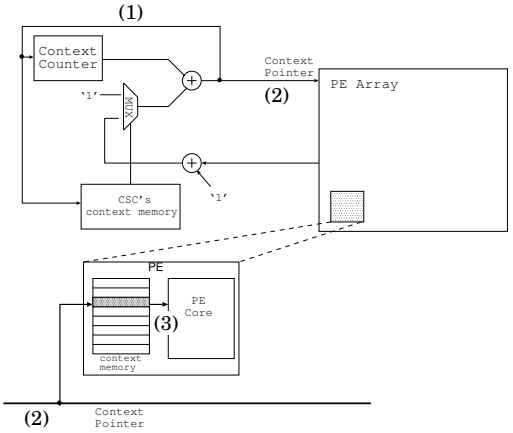


**Fig. 3**. Context control of MuCCRA-P

### 2.4. Context Switching and Task Control

In MuCCRA-P, 32 hardware context are controlled by CSC (Context Switching Controller) which uses a simple context counter. Reconfigurable modules including PEs and SWs load configuration data from context memory according to a context pointer generated from the context counter in the CSC. It is also reconfigurable module, and the configuration data for context control is also loaded for itself.

The context counter is simply incremented when the branch is not specified or not taken in the context. In MuCCRA-P, the branch address and branch condition signals are computed in the PE array and sent from the special PE to the CSC. If the branch is taken, the branch address is added with the context pointer in the CSC. By using the mechanism, a table jump according to the computation results can be implemented as well as simple loop structures.

TCC (Task Configuration Controller) provides 1K depth central configuration memory for storing configuration data for each task, and the configuration code is multicast using

the bitmap[12] to the context memory modules in PEs and SEs before starting the task.

### 2.5. Area

MuCCRA-P was designed for a 5.00-mm square die in Aspla 90nm CMOS technology. The area of MuCCRA-P core is almost 4mm $\times$ 4mm. The RTL model is described in Verilog-HDL. Synopsys Design Compiler 2006.06-SP2 and Synopsys Astro 2007.03-SP3 are used for logic synthesis and layout, respectively. Note that the standard clock gating supported by Design Compiler is applied. That is, clock is stopped when the module is not available.

Table 3 (in Section 4) shows detail numbers of cell area. See the column "MuCCRA-P" for the original MuCCRA-P. Area of a PE module except context memory accounts for 32% of total area, context memories and CSC module for reconfiguration is about 20%. The ratio of context memories are 37% on PEs and 43% with each other.

## 3. POWER ANALYSIS OF MUCCRA-P

### 3.1. Power Classification on MuCCRA-P

First, in order to analyze power consumption on an architectural level, we classified power as shown below.

- *Processing power* is consumed in arithmetic components in PEs; ALUs, SMUs and output of RFiles. The power for distributed memory module is shown as "MEM".
- *Interconnection power* is for data communication between PEs or PEs and MEMs through SEs.
- *Reconfiguration power* is total power for the context switching. As shown in Figure 3, the context switching is performed as follows: first, the context pointer for the next context is computed in CSC and broadcasted to the context memory modules. Then, the configuration data is read out from the context memory, and the functions in PEs and interconnects are changed. Thus, reconfiguration power is consisting of three components: 1) power consumed in CSC, 2) power for read and set the configuration power, and 3) clock and leakage power for the context memory. The power for changing the datapath on the PE array is included in *processing power*.
- *Standby power* includes power for clock distribution and leakage in the total PE array except the context switching mechanism.

### 3.2. Power Analysis

#### *3.2.1. Evaluation environment*

For evaluation, we used four programs: discrete cosine transform (DCT) used in JPEG coder, a simple image processing program called alpha blender (Alpha), a hash function used in encryption (SHA-1) and discrete Wavelet transform (DWT) used for data compression. All of them were described in a C-like hardware description language and implemented using the Black-Diamond retargetable compiler[13]. For optimization, pragma description to specify the location of PEs is used. The resources used in each application are shown in Table 1.

**Table 1**. Resources used in applications

| | Context | Cycle | ALU | SMU | RFU | PE |
|---|---|---|---|---|---|---|
| DCT | 29 | 99 | 567<br>35% | 385<br>24% | 1152<br>72 % | 1208<br>76% |
| SHA-1 | 8 | 490 | 2920<br>37% | 1633<br>21% | 1604<br>71 % | 6018<br>77% |
| DWT | 8 | 136 | 1046<br>48% | 660<br>30% | 538<br>25% | 1256<br>58% |
| ALPHA | 6 | 15 | 82<br>34% | 98<br>41% | 92<br>38% | 146<br>61% |

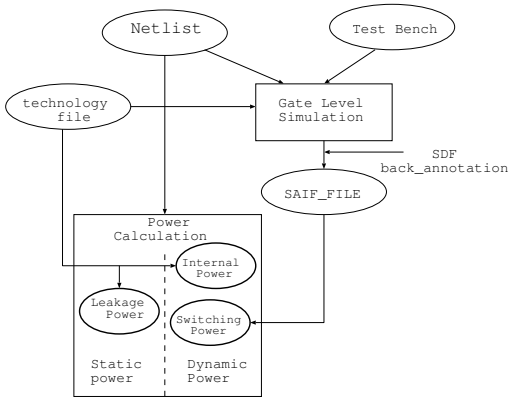


**Fig. 4**. The flow of power analysis

The flow of analyzing the power consumption is shown in Fig. 4. First, MuCCRA-P post-layout netlist with back annotation data is simulated using the configuration data set for each application. Then, Synopsys's PrimePower is applied to analyze the consuming power.

Here, all applications are assumed to run with 33MHz clock. The power for I/O and loading configuration data to the configuration memory is omitted.

### 3.3. Evaluation results

#### *3.3.1. Power consumption of each application*

First of all, power consumption with the above classification is shown in Fig 8 (This figure is placed in Section 4 with the structure with power reduction techniques. The powers of MuCCRA-P without power reduction techniques are shown in the left most position.) The total power consumption is less than 60mW. The performance of each application is from 2-6 times as that of TI's DSP TMS320C6713 which works at 225MHz clock. Since the consuming power of the catalog specification of TMS320C6713 is 1.1W, the perfor-

mance per energy of MuCCRA-P is much better than the DSP.

For all applications, the most power consuming component is *processing power*(note that, the power for distributed memory modules are shown as "MEM" in the graph), but the ratio is depending on the utilization of PEs. It becomes about 50% in DCT that requires a lot of PEs in each context, while it is small in SHA-1 which is difficult to utilize a lot of PEs in a context. Unlike FPGAs, *interconnect power* is about 17%, and not dominant in the total power consumption.

*Reconfiguration power* consumes about 15mW independent of applications. It occupies about 20%, and is not a dominant factor in the total power consumption. Considering that contexts are switched almost every clock in applications implemented here, the claim that the power for context switching dominants the multicontext DRPA is just a myth. The power required for changing datapath dynamically will be analyzed lator. *Standby power* also consumes about 10mW independent of applications. Although the clock gating is applied, the power is mainly consumed in the clock tree, and it is difficult to be reduced without using special clock distribution method.

The analysis results suggest that the power consumption by the data-path for computation formed on the PE array is dominant factor, and so the power reduction techniques for the data-path in common microprocessors would be also useful in MuCCRA-P.

#### *3.3.2. Power for changing datapath dynamically*

From the above evaluation results, the power consumption for context switching itself; that is, managing context control in CSC, reading and setting configuration data from the context memory, clock for context memory and leakage of hardware amount for context management; is about 20% of the total power. However, it does not include the power for changing datapath on the PE array dynamically. In general, just after changing the datapath by switching the configuration data, the computation starts continuously in the same clock cycle. So, it is difficult to separate the power for changing the datapath from the computation on the PE array. In order to separate the effect of switching datapath, the following three cases were evaluated.

Case 1: no context switch. A computation is done continuously in the same context without context switching (Figure 5 (1)) .

Case 2: context is switched but the datapath is not stitched. Context A and Context B executes exactly the same computation with the same mapping, and are switched every clock cycle (Figure 5 (2)) .

Case 3: both context and datapath are switched. Context A and Context C executes exactly the same computation but the mapping is different. They are switched every clock cycle (Figure 5 (3)) . Here, the datapath is simple total sum with constant values with 8 PEs, and the clock frequency is also set to be 30MHz.

Table 2 shows the result of power for each case. The difference between Case 1 and Case 2 comes from *reconfiguration power*. Although context switching is not performed in Case 1, the power for CSC and clock distribution to the context memory are required, and configuration pattern is differnt from Case 1. On the other hand, the difference between Case 2 and Case 3 comes from changing datapath. The evaluation results show that the power by changing the datapath is not so large compared with *reconfiguration power*.

**Table 2**. Reconfiguration power match-up of test pattern

| Test Pattern | Power [mW] |
|---|---|
| Case 1 | 28.2 |
| Case 2 | 41.3 |
| Case 3 | 44.0 |

## 4. POWER REDUCTION TECHNIQUES

Here, we tried to apply to power reduction techniques based on the analysis results.

### 4.1. Operand Isolation in PE

The analysis results suggest that conventional power reduction techniques for microprocessor datapath will work efficiently. Operand isolation which fixes the input of functional units is typical power reduction technique for microprocessor datapath. SMU and ALU in a PE provide 32 functional units in total. Although some units share resources, most units work only for their own operation, and the power consumption by unnecessary output toggles can be reduced by operand isolation. In this mechanism, each instruction unit has AND units to choose the input data from PICKIN or fix data '0'.

Fig. 6 shows operand isolation for the ALU. A simple decoder is provided and gives '1' to an input of AND units for a selected unit, thus, the inputs of other units are fixed to data '0'. By using the mechanism, the outputs of unused

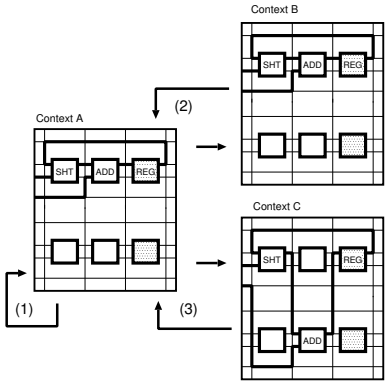


**Fig. 5**. Three cases to analyze the context switching

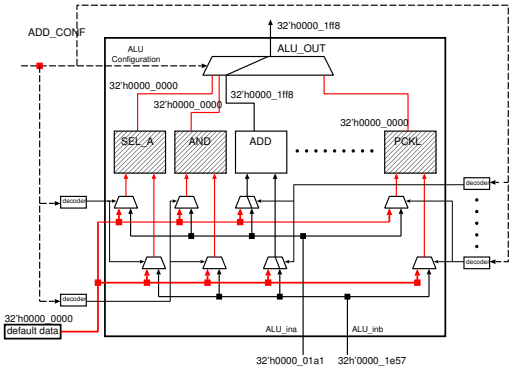


**Fig. 6**. Operand isolation and toggle action

PEs are also fixed by fixing their inputs, we don't have to provide buffer to fix the output data.

### 4.2. Selective Context Fetch

Analysis results also suggest that a large part of power consumption for context switching comes from the reading configuration data from the context memory. Although MuCCRA-P has 16 PEs and 25 SEs, all of them are not used in a certain context. For such unused units, the context memory can be in the stand-by mode by disabling CE (Chip Enable), and default configuration data is replaced with the data from the context memory.

In this mechanism, which is called selective context fetch, as shown in Fig 7, all PE and SW module have flag registers corresponding to the context depth which indicates whether it will be used in the context or not. These flags are treated as a part of configuration data, and transferred from configuration memory in TCC in advance. When the context is changed according to the context pointer, the flag is checked, and only required context memory is enabled and the configuration data is read out. Otherwise, the default configuration data is used. Since it takes a clock to read the context memory as common synchronous memory, the delay for checking flag must be added to the latency. However, in MuCCRA-P, the access time is much smaller than the delay for the datapath on PE array, and the operational frequency is not influenced.

### 4.3. Evaluation of power reduction techniques

Both techniques increase the hardware which will lower the effect of power reduction. Table 3 shows the number of cells of each module of the original MuCCRA-P, one with the operand isolation (MuCCRA-OI) and one with both the operand isolation and selective context fetch (MuCCRA-SF). Note that figures in the table come from the layout and includes buffers for clock tree and timing ajustment to fix the hold time error.

In MuCCRA-OI, small (about 3%) overhead of area by additional logic provided at inputs of each unit is required. On the contrary, in MuCCRA-SF, a considerable overhead

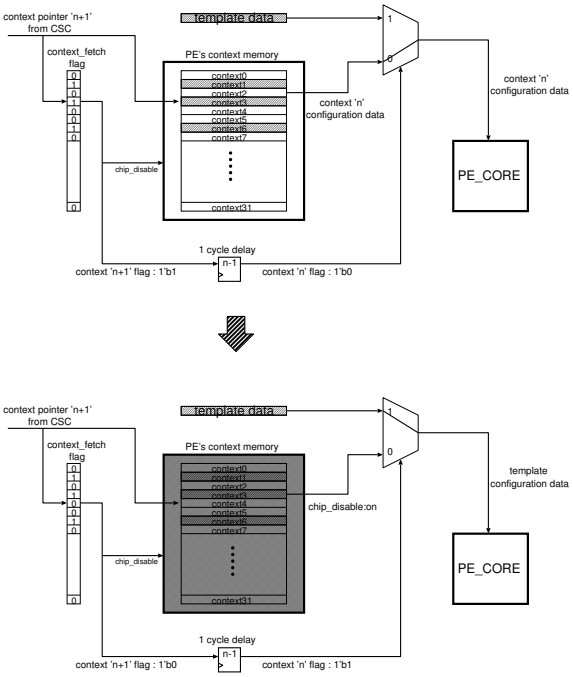


**Fig. 7**. Selective context fetch

**Table 3**. Cell area($\mu m^2$)

| Module | MuCCRA-P | MuCCRA-OI | MuCCRA-SF |
|---|---|---|---|
| PE $\times$16 | 935,328 | 988,672 | 1,116,512 |
| PE_CMEM $\times$16 | 351,136 | 351,136 | 351,136 |
| SE $\times$25 | 283,650 | 283,125 | 379,625 |
| SE_CMEM $\times$25 | 219,075 | 219,075 | 219,075 |
| MEM $\times$4 | 697,364 | 658,784 | 707,392 |
| MEM_CMEM | 34,466 | 34,466 | 34,466 |
| CSC | 11,618 | 11,619 | 12,072 |
| etc | 243,538 | 212,872 | 252,844 |

(15% in PE, 34% in SE) is required by an array of flip-flops for flags, multiplexers for addressing, and additional buffers for forming the clock tree.

Figure 8 shows the power consumption of three structures (MuCCRA-P, MuCCRA-OI and MuCCRA-SF) using four applications. The operand isolation reduces the power from 30% to 40% of *processing power* and 10%-20% in total. On the other hand, the selective context switch reduces *reconfiguration power*, but the impact to the total power is less than 8%. Two reasons degrade the power reduction effect by the selective context switch: (1) the large additional hardware for flags requires stand-by power and static power, and (2) if the implementation of algorithm is well optimized, not so many PEs and SEs can stop fetching the configuration data. Table 4 shows average numbers of PEs and SEs which can stop fetching the configuration in a datapath. It shows that the total numbers are not so large in common applications.

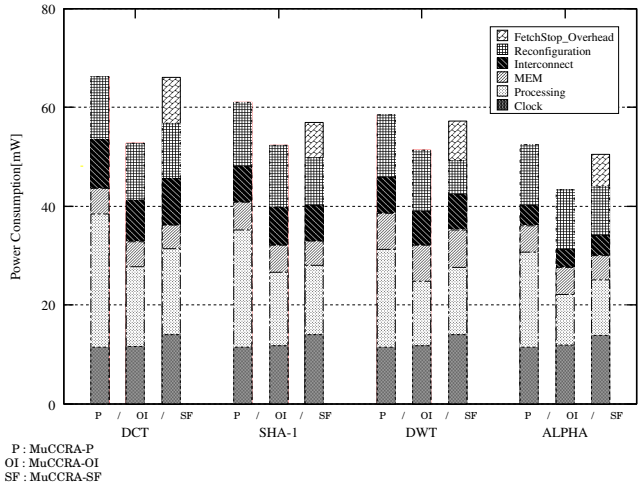


**Fig. 8**. Power consumption of three structures

**Table 4**. Average numbers of units which can stop fetching

| Application | PE | SE | Reduction [mW] |
|---|---|---|---|
| DCT | 0.2 | 3.2 | 0.1 |
| SHA-1 | 3.8 | 7.9 | 3.2 |
| DWT | 6.5 | 12.2 | 5.1 |
| ALPHA | 6.0 | 6.4 | 1.7 |

Now, let's verify the possible improvement of the selective context fetch. When it negates the chip enable of context memory modules, dynamic power can be reduced up to 90%. Let the power consumed in the context memories be $P_{cmem}$, power of the hardware of selective context switch be $P_{overhead}$, and the applicable ratio of the selective context fetch be $n$, respectively. The break-even point of power gain with the selective context switch is represented as follows:

$$P_{cmem} * 0.9 * n \geq P_{overhead}$$

From the evaluation, the $P_{overhead}$ becomes 2.5[mW] in static power, and 8.1[mW] in dynamic power. The result of DWT shows that the power of context memories for PEs is 9.2[mW] and those for SEs are 2.4[mW], Thus, the formula can be rewritten as follows.

$$(9.2 + 2.4) * 0.9 * n \geq 2.5 + 8.1$$

From this formula, $n$ must be at least 77% in order to achieve power reduction. Considering the results from Table 4, it is difficult to reduce the power by using the selective context fetch in this architecture and resource utilization.

## 5. CONCLUSION

The power consumption of DRPA is quantitatively analyzed, and two dynamic power reduction techniques; functional unit-level operand isolation and selective context fetch are proposed. Evaluation results demonstrate that the functional unit-level operand isolation can reduce up to 20.8% of the dynamic power with only 2.2% area overhead. On the selective context fetch, the power reduction is limited by the increasing of the additional hardware.

**Acknowledgments:** This work is supported in part by Japan Science and Technology Agency (JST). The authors thank to VLSI Design and Education Center (VDEC).

## 6. REFERENCES

[1] F. Veredas, M. Scheppler, W. Moffat, and B. Mei, "Custom Implementation of the Coarse-Grained Reconfigurable ADRES Architecture for Multimedia Purposes," in *Proc. of FPL*, Aug. 2005, pp. 106–111.

[2] M. Motomura, "A Dynamically Reconfigurable Processor Architecture," *Microprocessor Forum*, Oct. 2002.

[3] T. Sugawara, K. Ide, and T. Sato, "Dynamically Reconfigurable Processor Implemented with IPFlex's DAPDNA Technology," *IEICE Trans. on Information & System*, vol. E87-D, no. 8, pp. 1997–2003, May 2004.

[4] M. Petrov, et al., "The XPP Architecture and Its Cosimulation within the Simulink Environment," in *Proc. of FPL*, Aug. 2004, pp. 761–770.

[5] Rapport, Inc., http://www.rapportincorporated.com/.

[6] T. Kodama, et al., "Flexible Engine: A Dynamic Reconfigurable Accelerator with High Performance and Low Power Consumption," in *Proc. of Int'l Symp. on Low-Power and High-Speed Chips (COOL Chips)*, Apr. 2006, pp. 393–408.

[7] T. Stansfield, "Using Multiplexers for Control and Data in D-Fabrix," in *Proc. of Int'l Conf. on Field Programmable Logic and Application (FPL)*, Sept. 2003, pp. 416–425.

[8] Y. Kurose, et al., "A 90nm Embedded DRAM Single Chip LSI with a 3D Graphics, H.264 Codec Engine, and a Reconfigurable Processor," in *Hot Chips 16*, Sept. 2004.

[9] T.Nishimura, K.Hirai, S.Takeda, Y. Hasegawa, S. TsuTsumi, K.Usami, and H. Amano, "Power Reduction Technique for Dynamically Reconfigurable Processor," in *Proc. of the 10th IEEE Symp. on Low-Power and High Speed Chips (COOL Chips X)*, April 2007, p. poster session.

[10] T. Y.Hasegawa, S.Tsutsumi, and H.Amano, "Power analysis on Dynamic Reconfigurable Processor," in *IEICE Technical Reports (DESIGN GAIA2007)*, November 2006, pp. 31–36, (In Japanese).

[11] H.Amano, Y.Hasegawa, S.Tsutsumi, T.Nakamura, T.Nisimura, V.Tanbunheng, A. Parimala, T.Sano, and M.Kato, "MuCCRA Chips: Configurable Dynamically-Reconfigurable Processors," in *Proc. of ASSCC 2007*, Nov. 2007, pp. 384–387.

[12] V. Tanbunheng, M. Suzuki, and H. Amano, "RoMultiC: Fast and Simple Configuration Data Multicasting Scheme for Coarse Grain Rec onfigurable Devices," in *Proc. of FPT*, Dec. 2005, pp. 129–136.

[13] V. Tunbunheng and H. Amano, "Black-Diamond: a Retargetable Compiler using Graph with Configuration Bits for Dynamically Reconfigura ble Architectures," in *Proc. of The 14th Workshop on Synthesis And System Integration of Mixed Information technologies ( SASIMI)*, 2007, pp. 412–419.