A Circuit Design Method for Dynamic Reconfigurable Circuits

Hajime Sawano†, Takashi Kambe‡
†Graduate School of Science and Engineering, Kinki University
3-4-1 Kowakae, Higashi-Osaka City, Osaka, Japan
‡Depart. of Electric and Electronic Engineering, Kinki University

Abstract— Reconfigurable Computing (RC) is a new paradigm that addresses the conflicting design requirements of high performance and high area density. In Coarse Grained Architecture (CGA) RC systems, it is important to achieve acceleration using pipelining and also achieve a high Processing Element(PE) utilization ratio. This paper proposes an interactive circuit design methodology for Dynamically Reconfigurable Processors to accelerate their performance and achieve compact, low power circuits. The method is applied to a JPEG encoder design and its performance evaluated.

I. Introduction

Reconfigurable Computing (RC) is a new paradigm that addresses the conflicting design requirements of high performance and high area density. Many fine-grained and coarse-grained reconfigurable processor architectures have been proposed [1, 2]. Coarse-grained architectures (DAPDNA[9], DRP[5]) operate at the word level of granularity and exhibit better power and performance than fine-grained architectures. The Processing Element(PE) of coarse-grained architecture can perform multiple operations with multiple arithmetic units.

In a CGA-RC system, the key issue is to take full advantage of the multifunctional features of the PE for all applications. Therefore, it is important to obtain both acceleration by pipelining and high PE utilization.

Recently, several automatic design approaches for RC system have been developed [5, 6, 7, 8]. However, interactive design is the still the main approach currently used. Designs for a variety of applications have been reported, but there is little detailed discussion about the design method. Here, we propose an interactive GUI-based design method for the DAPDNA-2 and evaluate its performance by applying it to a JPEG encoder design.

II. Dynamic Reconfigurable Processor Architecture

The DAPDNA-2 is a RC device that consists of two processing cores. One core is the Digital Application Processor (DAP), which is a custom-designed 32-bit RISC processor. The DAP core performs system-control and handles general purpose operations. The other core is the Distributed Network Architecture (DNA) core which is a two dimensional array of 32-bit dynamically reconfigurable PEs. The operating frequency of each processing core is 166MHz.

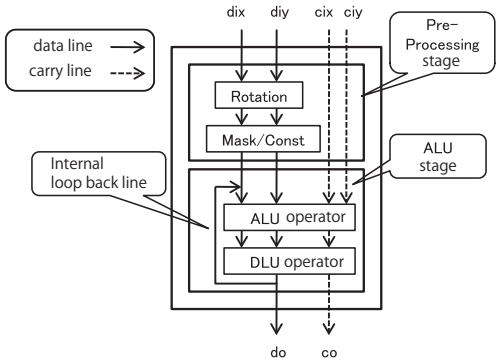

Fig. 1. EXE Element

The behavior of each PE and the connectivity among PEs are defined by DNA configurations which can be switched in one clock cycle. The DAPDNA-2 can store up to four configurations internally; one in the foreground memory bank and three in the background. These banks can be loaded with new configurations from external memory thus providing an unlimited number of configurations.

The DAPDNA-2 includes several types of PE that support many kinds of processing. Table I lists the types of PE that are available. An EXE element performs multiple arithmetic operations and has a major role in pipelining. Fig. 1 shows the internal structure of an EXE element. An EXE element consists of a Pre-Processing (PrePro) stage and an ALU stage. The internal loopback line enables the result of an ALU stage calculation to be input into the same ALU stage again in the following cycle. This can be used when there is a data dependency in the loop structure.

TABLE I
PROCESSING ELEMENTS

Name	Qty	Function
EXE	168	Arithmetic and logic operations
Delay	136	Variable delay for synchronization
RAM	32	Internal temporary memory
Counter	24	General-purpose counter
LDB/LDX	8	Data input channel to DNA
STB/STX	8	Data output channel from DNA

III. Interactive Design method for RC system

The following are the key design style points for high-speed, large-scale data processing circuits in the DAPDNA-2. Other RC systems adopt similar design styles.

1. Implement parallel processing and/or pipelining circuits in a single configuration.
2. When several configurations are used for large scale designs, store the configuration data in the memory bank and then load each configuration using the autonomous-switching mechanism. It takes only one clock cycle and enables high speed processing.
3. When large volumes of data are processed, store the data in external memory and access it from PE using the internal RAM buffers.

We propose the following design method to achieve short design time and low power consumption.

1. If the PE utilization rate of each configuration is less than 60%, then the automatic fitting tool provided by DAPDNA development group can be used. Otherwise the automatic fitting tool often fails in the DAPDNA-2 environment, but this is general issue in RC systems. Manual PE placement and routing takes a long time and the designer needs to understand all PEs and their routing structure in detail. Further, if logic has needs to be added to change the specification, its modification is difficult.

   However, when the PE utilization rate of each configuration is kept under 60%, more configurations and switching are sometimes required. This creates a small processing speed overhead, but the layout design time is shorter and the modification of circuit design is easier.
2. When large volume data is processed, the data is transferred between configurations using internal RAM. External memory is accessed as little as possible because this can reduce power consumption.

In case of JPEG encoder design, because of the limitation of internal RAM capacity, each configuration processes 80 MCU (Minimum Coded Unit) at a time and this is repeated to transform any size of picture.

IV. JPEG Encoder Design

The JPEG encoding process consists of MCU splitting, color space conversion sequence, discrete cosine transform (DCT), transposition, quantization, zigzag sequence, and Huffman coding. This section, describes the design of each function block and DNA configuration.

A. The design of each function block

1. MCU Splitting
   In this stage, the input image is split into MCU blocks and encoding is performed for each block. MCU splitting is implemented using main memory addressing. Because main memory access is slow, burst data transfer is used to realize high speed. The burst transfer data size for main memory access is up to 64 words. The size of one MCU is 16 words and the continuous burst transfer unit is 16 pixels. So we do double buffering using LDB elements.
2. Color Space Conversion
   Because MCU consists of four Y, one Cb, and one Cr blocks (8×8 pixels) and these blocks have no data dependencies, color space conversion is performed using six parallel processes.
3. DCT
   Two-dimensional DCT is implemented using one-dimensional DCT in the X and Y directions and using the Chen algorithm. The one-dimensional DCT is implemented using PE based parallel processing for eight columns with eight input on-chip RAMs.
4. Transposition
   Transposition is performed after the X direction one-dimensional DCT. This circuit consists of eight one-column based transposition circuits. In the one-column transposition circuit, each 32-bit Delay and EXE element handles two pixels simultaneously to convert from column to row (Fig. 2).
5. Quantization
   Quantization divides the output of the DCT operation using a quantization table. However, the division requires many PE and is slow. To address this, quantization is implemented using integer-based reciprocal multiplication with a shift operation and processes 8 pixels in parallel.
6. Zigzag Sequencing
   This process reads data from main memory and places them in internal RAM in zigzag sequence for

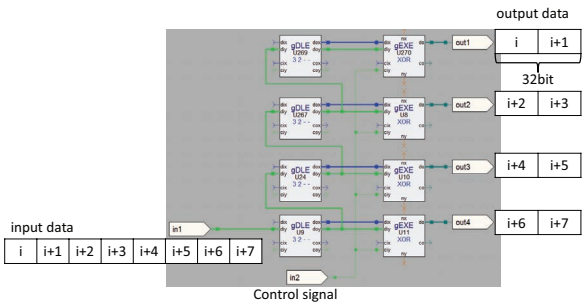

Fig. 2. Transposition

Huffman coding. To speed this up, double buffering using two internal RAM is used to read and write memory in parallel. In addition, these two circuits are implemented for parallel zigzag sequence processing.

7. Huffman Coding
   This circuit processes DC and AC Huffman coding in parallel. Because the AC components do not have data dependency on each other, two AC components are processed in parallel. Zero run length counting is performed using the EXE internal loopback.

B. *The DNA Configuration*

Fig. 4 shows the DNA processing flow for the JPEG encoder. The design has six configurations from MCU Splitting to Huffman Coding. Fig. 3 shows a design result for the fourth configuration - Zigzag sequence and Huffman coding.

MCU Splitting and Color Space Conversion share external memory accesses and are implemented in the same configuration. Because of the limitation on internal RAM capacity, the proposed DNA configurations can only handle 80 MCU. However, we can support unlimited image sizes by repeating this DNA configuration.

V. Design Results and their Evaluation

Table II shows the design results for the DNA configurations. The automatic fitting tool was able to fit all configurations into the DNA matrix.

At first, we designed configuration B. After that, we designed configuration A by improving configuration B. Two kinds of configuration A and B are shown in Table III . Configuration A has higher degree of parallelization, better memory access, better PE optimization, and higher speed configuration switching than configuration B.

Table IV shows a comparison between two CGA-RC design results and a JPEG software implementation. The input data are five 4096 × 3072 natural images. The table

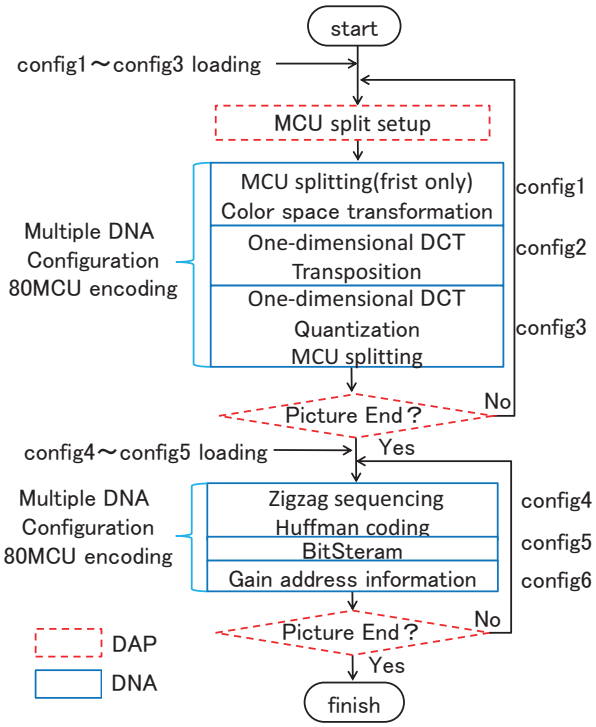

Fig. 4. DNA configuration

shows the processing time averages. All the compression ratios in the experiment were 98.5%. The JPEG software is written in C and based on [11]. This software uses fixed-point calculation and Chen algorithm. C compiler is gcc 4.6.1 (no options). The processing time was measured in a single thread, 3.16GHz, Xeon X5460. This table shows the processing time for 80 MCU encoding.

Config A is about 5 times faster than Config B and about 4.5 times faster than the software version. The Config B overhead is mainly due to the processing time of the DAP-controlled configuration switching. Even if autonomous switching is applied to Config B, Config A is

TABLE II
Configuration List

Config	EXE	Delay	Count.	RAM	LDB /STB	PE total
1	117	49	14	16	1/0	197
2	111	53	3	12	0	179
3	105	42	10	18	1/1	177
4	149	60	12	24	1/1	245
5	125	61	9	7	1/1	204
6	0	2	3	1	0/1	7

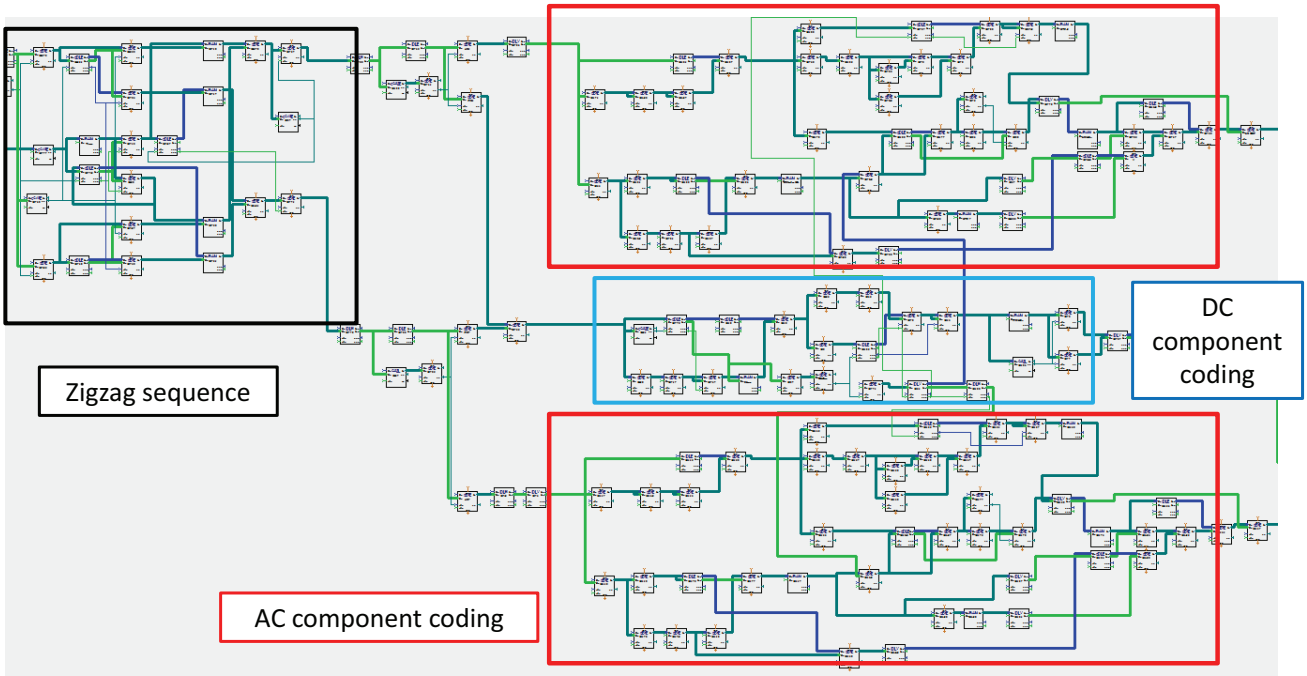

Fig. 3. Design Results for the 4th Configuration

TABLE III
TWO CONFIGURATIONS

Block	Config. A	Config. B
Burst transfer size of MCU split(words)	64	16
Parallel degree of transposition	8	4
Memory access of quantization	main memory	on-chip RAM
Parallel degree of Huffman coding	2	1
No. of configurations	6	8
Config. switching	autonomous	DAP control

TABLE IV
TWO DESIGN RESULTS AND SOFTWARE (μSEC)

	Config A	Config B		Software
MCUsplit	61	146		
color trans	33	61		389.2
X-DCT		24		
Trans.	24	47	71	201
Y-DCT				
QZN	(48)	24		229.4
Huffman	97	187		441
BitStream	64.6	64.6		49.8
overhead	12	896		-
total	291.6	1449.6		1310.4

still about 2 times faster because it utilizes more parallel processing and on-chip memory.

VI. CONCLUSION

We have proposed an interactive design method for RC-systems in section III and described the design of an JPEG encoder in section IV. Circuit speed is increased by improving the PE utilization efficiency and adopting parallel PE-based pipelining by comparing the configuration A and B in section V.

The circuit can compress any size of image by repeating the 80 MCU based process. This investigation has enable us to confirm the effectiveness of our design approach.

Based on this research, we plan to develop the algorithm further to generate DNA configurations automati-

cally and evaluate it in a wide range of applications.

References

[1] Reiner Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in Proc.of DATE, 2001.

[2] Jozwiak, L., N.Nedjah, et al."Modern development methods and tools for embedded reconfigurable systems: A survey." Integration, the VLSI Journal 43(1): pp1-33(2010).

[3] H.Amano, "Recent trends on Reconfigurable/Dynamically Reconfigurable Systems [in Japanese],"IEICE technical report 105(36), 31-36, 2005, (in Japanese).

[4] K.Okada, et al., "Hardware Algorithm Optimization Using Bach C," IEICE Trans. Fundamentals vol.E85-A, No.4, pp835-841, 2002.

[5] Takao, T., N.Noritsugu, et al.:"High-level synthesis challenges and solutions for a dynamically reconfigurable processor," Proceedings of the international conference on Computer-aided design.San Jose, California, ACM, (2006).

[6] Jozwiak, L.and A.Douglas: "Hardware Synthesis for Reconfigurable Heterogeneous Pipelined Accelerators," Information Technology: New Generations, Fifth International Conference on, (2008).

[7] Wolinski, C., K.Kuchcinski, et al.: "Automatic design of application-specific reconfigurable processor extensions with UPaK synthesis kernel." ACM Trans.Des.Autom.Electron.Syst.15(1): pp1-36, (2009).

[8] Xydis, S., K.Pekmestzi, et al.:"High-Level Synthesis Methodologies for Delay-Area Optimized Coarse-Grained Reconfigurable Coprocessor Architectures.VLSI (ISVLSI), IEEE Computer Society Annual Symposium on, (2010).

[9] Sugawara, T., et al.:"Dynamically Reconfigurable Processor Implemented with IPFlex's DAPDNA Technology." IEICE Transactions 87-D(8): pp.1997-2003, (2004).

[10] Wen-Hsiung Chen, Smith C., Fralick S.:"A Fast Computational Algorithm for the Discrete Cosine Transform," Communications, IEEE Transactions on ,vol.25, no.9, pp.1004- 1009, Sep 1977.

[11] H. Sawano, et al. :"JPEG encoder design and its evaluation for Dynamic Reconfigurable Circuit," vol.132, No.11, Sec.C, IEEJ Transactions on EIS, 2012, (in Japanese).