Journal of Information Processing Vol.21 No.3 552–562 (July 2013)

[DOI: 10.2197/ipsjjip.21.552]

Regular Paper

Flexible and High Performance ASIPs for Pixel Level Image Processing and Two Dimensional Image Processing

Hsuan-Chun Liao[1,a)] Mochamad Asri[1,b)] Tsuyoshi Isshiki[1,c)]
Dongju Li[1,d)] Hiroaki Kunieda[1,e)]

Received: November 3, 2012, Accepted: April 5, 2013

Abstract: An image processing engine is an important component in generating high quality images in video systems. Processing during capture and display are non-standard and vary from case by case, hence, the flexibility of image processing engines has turned out to be an important issue. The conventional hardware type of image processing engine such as an Application Specific Integrated Circuit (ASIC) is not applicable for this case. In order to increase design reusability and ease time-to-market pressures, Application Specific Instruction-set Processors (ASIP) which provide high flexibility and high computational efficiency have emerged as a promising solution. In this paper, we present two ASIPs. PXL ASIP, which has a reconfigurable multi bank memory module and an SIMD type computation pipeline, is designed for pixel level image processing, while 2D ASIP, which has slide register module and reconfigurable ALU modules, is designed for 2D image processing. PXL ASIP can perform 4 to 10 times faster compared to its base processor, and 2D ASIP can perform 5 to 43 times faster compared to its base processor.

Keywords: image processing, reconfigurable processor

1. Introduction

Pre-processing after capturing images from image sensors, post-processing before outputting image to display device, and coding and decoding (CODEC) to compress and decompress images are three main types of processing in modern video systems. **Figure 1** shows the processing block diagram of general video systems. In order to perform the different types of processing, often ASICs are developed for different applications. However, this is a time-consuming and costly solution. In order to ease the time-to-market pressures and to maximize design reusability, programmable or reconfigurable hardware has been gradually proposed [1], [2]. The variety of processing is one of the issues in the implementation of image processing engines, and performance requirements turn out to be another issue. In recent years, the resolution of video systems is becoming higher from 640 × 486 (NTSC) to 1,920 × 1,080 (Full HD). Some cameras and LCD TVs have even started to support 4,096 × 2,160 (4K × 2K) resolution. In the Full HD case, if the working frequency of the processor is 400 MHz, the allowable processing time for each pixel is 6.43 cycles (400 M/(1,920 × 1,080 × 30)). Uniprocessor architecture is not efficient in supporting such a high performance requirement; therefore, multiprocessor-based architectures are being adopted [3], [4], [5] to support high computational image/video processing.

In designing a high performance multiprocessor architecture for the image processing engine, the capability of processing element and communication architecture which is used to connect the processing elements are essential issues. The common hardware types of processing element on the market can be classified into ASIC, reconfigurable ASIC, Application specific instruction-set processor (ASIP), DSP and general purpose processor (GPP). ASIC is area and power optimized for a specific application. The state-of-the-art GPP has very complicated branch predictor and multi-issue ALU. It has the best flexibility, but computational efficiency is the worst against other hardware types. DSP usually has an independent data memory and multiply-accumulate logics to improve performance for computing-intensive applications. ASIP falls in between reconfigurable ASIC and DSP. ASIPs are simple processors optimized for specific applications. They combine the characteristics of higher computational efficiency and some extent of flexibility.

Some reconfigurable architectures can also provide programmability. Fine-grained reconfigurable architectures such as Field-Programmable Gate Arrays (FPGAs) consist of many bit-level logic elements and configurable interconnections. FPGA can provide the best flexibility, but the complex interconnection

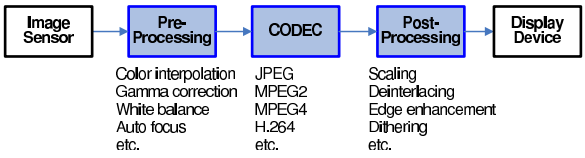

Fig. 1 Block diagram of video systems.

[1] Department of Communications and Integrated Systems, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[a)] liao@vlsi.ss.titech.ac.jp
[b)] asri@vlsi.ss.titech.ac.jp
[c)] isshiki@vlsi.ss.titech.ac.jp
[d)] dongju@vlsi.ss.titech.ac.jp
[e)] kinieda@vlsi.ss.titech.ac.jp

occupies too much chip area, and causes higher power consumption. Furthermore, the configuration time of FPGA is longer than the configuration time of processor, and this will bring to worse performance during switching algorithm. Coarse-grained reconfigurable platforms such as X4CP32 [12], and PARS [13] try to provide the best performance for various ranges of processing with the processing elements of a unified architecture. However, this design concept causes low area efficiency. ASIPs, on the other hand, which are optimized for image processing can provide required performance, besides excellent computational efficiency. We are targeting several specific image processing applications. In this case, ASICs and DSPs solutions are unfortunately inappropriate, in terms of flexibility and area-efficiency for targeted applications. ASICs can't cope with flexibility properly, while DSPs provide overly broad flexibility, sacrificing large area and power consumption. In this work, we are proposing a reconfigurable ASIP-based approach to accommodate both the flexibility and performance constraints, achieving near optimal trade-off between flexibility, performance and area.

In this paper, an image processing engine which consists of several kinds of ASIPs is presented. Each ASIP is designed for a specific domain of image processing, so a better computational efficiency can be provided, and the flexibility of ASIPs can support variety of image processing.

The rest of this paper is organized as follows. Section 2 presents a brief overview of related work. The characteristics of image processing algorithms in pre/post processing pipelines are explained in Section 3. In Section 4, we present the design process of our ASIPs. Section 5 describes the architecture of ASIPs. Section 6 explains the proposed parallel architecture of image processing engine. Section 7 summarizes the implementation results and comparisons. The conclusion and discussion are given in Section 8.

2. Related Work

There are several ways to implement processing elements in an image processing engine that can perform algorithms in both pre-processing and post-processing systems. The first solution is by reconfigurable ASIC. In paper [10], several kinds of reconfigurable stage processing elements (RSPE) are designed, trying to support algorithms which have similar characteristic of computation as much as possible. However, the datapath in each RSPE is fixed. User can only change the parameters and the sources of input data. The area of this type of hardware can be effectively reduced, but it cannot accommodate any additional computations (even addition) from the original designed datapath, which reduces its flexibility.

Another common method is by DSP-based solutions. DSPs which are commonly available in markets are mainly designed for communication and multimedia applications, and offer very good flexibility for image signal processing. Nevertheless, DSP has certain drawbacks as it requires larger area and higher power consumption than ASIC. Moreover, a powerful Very Long Instruction Word (VLIW) compiler which can utilize all the computational resources of DSP could provide positive impact on the performance of DSP. DSP-ASIC hybrid solutions [1], [11] offer another option to deal with this trade-off problem. By these solutions, ASIC parts are used to perform the well-defined processing and DSPs are used for nonstandard processing. It has the characteristics of both ASIC and DSP, so the designer must handle the functionality partition to attain the best performance. Reference [11] showed that when the number of supported applications increases to a certain amount, the area of dedicated HW (ASIC) starts to get larger than reconfigurable HW. In such situation, the advantage of dedicated HW gradually disappears. This fact fully matches our design condition, which is to design an engine which can support as large number of image processing applications as possible.

Some coarse-grained reconfigurable processors such as X4CP32 [12], and PARS [13] have been designed intending to provide the best performance for various range of processing with processing elements of a unified architecture. However, since each image processing has different characteristics, it is difficult to reach the best area or computational efficiency. In this work, we present a multiprocessor approach for realizing high performance image processing. The system architecture will be consisted of 1 dimensional (1D) ASIP, 2 dimensional (2D) ASIP and pixel level ASIP connected to a crossbar bus. In previous work [21], we proposed 1D ASIP and 2DX ASIP. However, while pixel level processing can perform moderately competitive performance, it cannot achieve Full HD performance requirement. Therefore, a pixel level ASIP is developed. Moreover, we also improve the performance of 2D ASIP, so that it can match state-of-the-art DSP video processor with much lesser chip area and power consumption.

3. Algorithms in Video System

Algorithms of CODEC are standardized in video systems. However, algorithms of pre/post processing have no explicit constraints and the effect of processing is hard to evaluate using objective factors. Therefore, ASICs which are optimized in performance and chip area for a specific application are hard to benefit from pre/post processing. Based on the above points, we think that image algorithms of pre/post processing are suitable for sharing hardware resources, and we can benefit from using ASIP instead of ASIC. In order to design a high performance ASIP, designer has to completely understand the characteristics of target applications. In this section, we use a pre-processing example and a post-processing example to roughly introduce the target applications.

A typical image processing pipeline for pre-processing is depicted in **Fig. 2**. Images are captured from the sensor in a Color Filter Array (CFA) format, typically Bayer pattern. Applications starts from auto white balance to false color suppression can be executed by ASIPs. Details of these applications in preprocessing pipeline are explained as follows.

Auto White Balance (AWB). The illumination during the recording of a picture is different from the illumination when viewing a picture. However, the white color level depends on illumination source. In AWB, the color histogram of an image is performed and analyzed to detect the source of illumination. The correct parameters of each color channel are used in the next

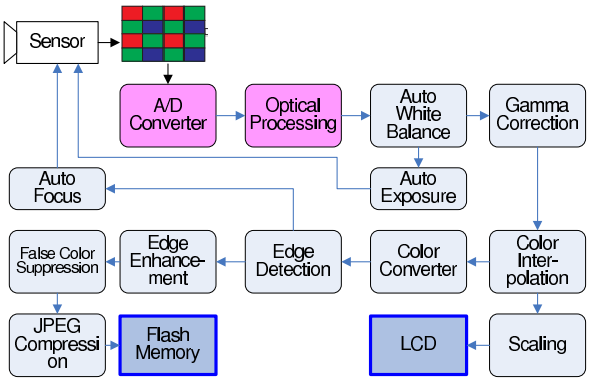

Fig. 2 The processing pipeline in normal DSC [1].

image.

Gamma Correction. Display devices can't accurately display colors as its input because the display property of display devices is usually nonlinear. Gamma correction is used to eliminate this problem. The common solution uses look-up table to convert pixel values.

Color Filter Array (CFA) Color Interpolation. The image sensor is covered by a color filter array (CFA), known as the Bayer array which is shown in Fig. 2. In the Bayer array, each pixel position only senses one color channel. This processing reconstructs the missing color information by interpolating neighboring pixels. The characteristics of color response are different in accordance with image sensors. The interpolation algorithm is also needed to be modified to fit the characteristic of image sensor.

Color Conversion. Typical image sensor operates on RGB color space, but image/video compression algorithms operate on YCbCr color space. A color space conversion is performed to transform the image from an RGB color space to a YCbCr color space.

Edge Detection/ Edge Enhancement. The nature of CFA color interpolation filters introduces a low-pass filter that smoothes the edges in the image. In order to enhance the details in an image, edge detection is used to compute the edge magnitude in the luminance channel at each pixel. The edge magnitude is then scaled and added to the original luminance value to enhance the sharpness of the image.

False Color Correction. Since the edge detection/edge enhancement are performed in the Y channel of the image, unprocessed chromatic channel introduces misalignment in the edge pixels color channels. False color correction suppresses the edge pixels chromatic information to reduce the above effect.

Following the information of pre-processing, a processing pipeline of the post-processing in HDTV is depicted in **Fig. 3**. Post-processing starts from the end of data decompression to the input of display device. Some applications such as color conversion and gamma correction are used on both processings. Some applications which are specific for post-processing are introduced as follows.

Deinterlacing. Interlace format was commonly used in analog television systems to save bandwidth. However, the display format in common display devices such as LCD is progressive. Deinterlacing interpolates the insufficient rows by referring to neighboring rows or neighboring frames.

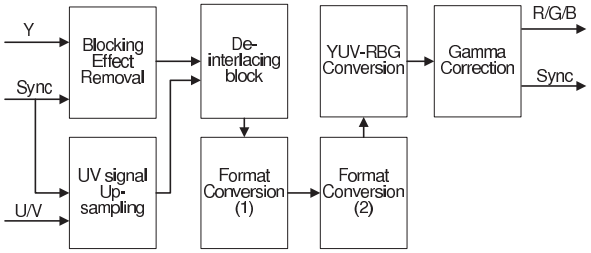

Fig. 3 The processing pipeline in normal HDTV [16].

Dithering. The color depth of display devices is restricted to the precision of D/A converter. To show deep color image on the low-end display devices, dithering adds an intentionally applied form of noise is used to randomize quantization error, so that the artifacts caused by the quantization error can be visually reduced.

4. Design Flow of ASIP

Our ASIP-based design flow starts from performance analysis in algorithm level. Then, the design flow is divided into hardware (HW) design flow and software (SW) design flow. HW design flow covers datapath design, then followed by instruction accurate processor design which is then transformed into a cycle accurate processor after the function verification of the algorithm. SW design flow starts with the performance analysis of a sequential application written in C. In order to figure out computational intensive operations in the application, we use tightly-coupled thread (TCT) instruction accurate simulator [17], TCT simulator has many evaluation tools which can help users to find out the hotspots in program and to predict the performance after adding special instructions. Two examples of datapath design and architecture design are explained as follows.

4.1 Design of Multi Bank Architecture for Pixel Level Image Processing

Pixel level image processing in pre/post processing such as color conversion doesn't need to refer to neighboring pixels, so processing performance can be improved by applying SIMD architecture. Several data is computed in parallel; therefore, a wider data memory (64 bit) [2], [18] is commonly adopted in SIMD architecture processors. However, the performance of processing which uses look-up table cannot be effectively improved by using SIMD architecture [18] due to the single I/O of the data memory. The memory which is constructed by multi banks seems to be able to solve this problem, but it fails when the accessed data are in the same bank. In order to solve the access collision problem, we did several simulations in different memory architectures and data allocations, and the probability of access collision is observed. In simulations, we accessed four elements in the table simultaneously, and we assumed the depth of each element in the look-up table is 8 bits. Therefore, a 32 bit word can contain 4 elements. The content of look-up table is assigned to memory word in raster scan order. **Table 1** shows the probability of access collision in different length of word in one memory. Although, the wider memory word can effectively reduce the collision rate, the collision rates are still too high to accept.

Table 1 Access collision rate in different length of word in one memory.

Image	Resolution	Word length			
		32 bit	64 bit	128 bit	256 bit
Foreman	176 × 144	87.13%	74.97%	59.48%	43.56%
Mobile	352 × 288	90.70%	83.37%	72.80%	60.50%
Ice	352 × 288	56.37%	43.29%	30.54%	22.00%
Station	1,920 × 1,080	86.52%	66.65%	42.84%	24.73%
Pedestrian	1,920 × 1,080	75.23%	55.25%	33.94%	17.96%

Table 2 Access collision rate in different number of bank of memory.

Image	Resolution	Bank number			
		4 banks	8 banks	16 banks	32 banks
Foreman	176 × 144	64.04%	28.79%	9.66%	2.63%
Mobile	352 × 288	70.70%	37.95%	16.83%	6.61%
Ice	352 × 288	28.17%	12.29%	4.72%	1.71%
Station	1,920 × 1,080	56.87%	18.12%	3.59%	0.47%
Pedestrian	1,920 × 1,080	46.55%	11.93%	2.05%	0.31%

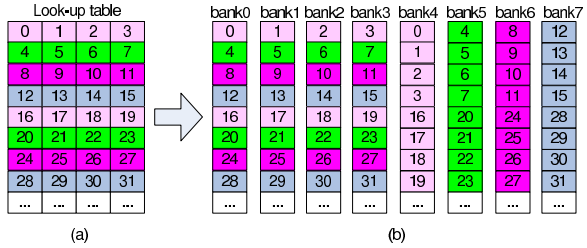

Fig. 4 (a) The content of look-up table. (b) The content after allocation in memory banks.

Table 2 shows the probability of access collision in different number of memory bank architectures. The width of each bank is 8 bit. The elements with index of a multiple of 4 in look-up table is assigned to bank0, and the elements with index of a multiple of 1 in the look-up table is assigned to bank1 and so on. As Compared to long memory word architecture, multi-bank architecture can further reduce the access collision rate. The decrement of the collision rates is lower than 10% in 32 bank case. However, the number of bank affects the chip area directly.

Based on the results of above simulations, multi bank is a feasible solution. We adopted 8 banks architecture as the basic memory architecture where we can improve the performance with lesser chip area. In order to further reduce access collision to about zero, we propose cross data allocation. **Figure 4** shows an example of the proposed data allocation. Elements in the first row of the look-up table are stored in the first position from bank0 to bank3 and position 1 to 4 in bank4. Elements in the second row of the look-up table are stored in the second position from bank0 to bank3 and position 1 to 4 in bank5.

Based on this allocation, 4 parallel access could achieve low collision, except for the case where the index of the parallel access is the multiple of 16. **Table 3** shows the collision rate of the proposed approach,which is 8-bank memory against the 32-bank memory. It shows the rate of access collision of the proposed approach is effectively suppressed to less than 1%, with only 8 memory banks are used.

4.2 Reconfigurable Datapath for 2D Image Processing

The target applications of 2D ASIP focus on 2D filtering operation such as edge detection, and smooth filtering. The characteristics of these 2D applications which include data input fashion, the type of computation and data output fashion are analyzed.

Table 3 Access collision rate in different number of bank of memory (the width of bank is 8 bit).

Image	Resolution	8 banks with cross allocation	32 banks with modulo allocation
Foreman	176 × 144	0.14%	2.63%
Mobile	352 × 288	0.49%	6.61%
Ice	352 × 288	0.13%	1.71%
Station	1,920 × 1,080	0.046%	0.47%
Pedestrian	1,920 × 1,080	0.033%	0.31%

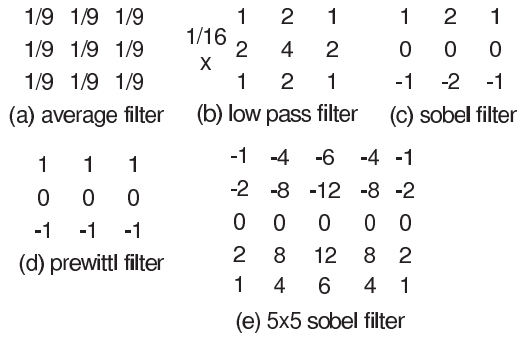

Fig. 5 The examples of 2D filter.

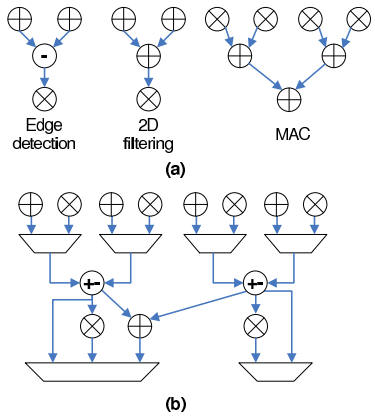

Fig. 6 The design of reconfigurable filter module. (a) The operations in target applications, (b) Merged circuit.

The similarity of the computation in applications can be utilized to share hardware resources. A design example of the reconfigurable filter module is shown in **Figs. 5** and **6**. First, we analyzed our 2D applications. Some examples of 2D image processing are shown in Fig. 5. We found normal 2D filtering (ex: low pass filtering) and edge-based filtering (ex: edge detection) have similar data access pattern and computation fashion. For example, the coefficients of filters are symmetry, and MAC operation is the core processing in applications. Therefore, the core operations are merged and the multiplexors are used to select the operation type. A special instruction with configuration information is used to activate the filter module and some relative logics. As 2D ASIP is designed based on a basic RISC processor, the applications which cannot be completely supported by reconfigurable modules can be performed by original pipeline of RISC processor.

5. Architecture of ASIPs

Processing elements in our image processing engine are implemented in ASIP. ASIPs are designed based on TCT ISA [19], which is a 4-stage pipelined, Harvard architecture 32-bit RISC

with local data and instruction memories. The four pipeline stages are instruction fetch (IF), instruction decode (DC), execution and memory access (EX), and (register or data memory) write back (WB). Communication modules are implemented by extending another stage (DS) after WB stage to accomplish the parallel processing with other ASIPs.

The proposed image processing engine consisted of three types of ASIPs in accordance with the types of processing. 1D ASIP is optimized for 1D filtering processing. The details is explained in Ref. [20]. In this paper, pixel ASIP and 2D ASIP are introduced. Pixel ASIP is optimized for applications which don't need to refer neighboring pixels. 2D ASIP is optimized for 2D filtering applications.

5.1 Architecture of PXL ASIP

Figure 7 shows the block diagram of PXL ASIP. Colored blocks are added into the basic RISC processor. The performance of ASIP for pixel level processing can be effectively improved by using SIMD architecture except for memory access operation. Therefore, A SIMD pipeline which can support four 16-bit computations and eight 8-bit computations is equipped to process most of the pixel level applications. 32-bit register file is used to handle data from basic 32-bit pipeline and 64-bit register files is used to handle data from SIMD pipeline. A permutation module is used to extract a part of data from a register, merge two registers and change data order in a register. An inner product module is implemented by reusing the arithmetic logics of SIMD datapath for inter channel operation such as color space conversion. In order to improve multi memory access operation which cannot be supported by normal SIMD architecture, we design a reconfigurable multi bank memory module.

Figure 8 is the reconfigurable multi bank memory module. Eight 1 KB memory banks are used to share look-up table, communication buffers, data storage and stack. Two kinds of addressing mode are supported and address conversion is handled by controller. The received data from other ASIPs are first stored in the receiver queue to avoid memory access conflict from the pipeline of ASIP at the same time. The memory access from the pipeline of ASIP has higher priority than the receiver queue. The address generation unit (AGU) is used to manage the addressing of communication data. The load and store unit (LSU) is used to handle the data access from the pipeline of ASIP. The content of look-up table is loaded with proposed cross data allocation before the operation of core processing. LUT4_Ex module in Fig. 7 is used to calculate the address of elements in memory, detect access conflict and select available banks. LUT4_Wb module reads data from memory banks and allocates data to the proper position in a register. If data fail to be accessed in one cycle, the rest of unavailable data will be accessed in the following cycles.

In order to support the parallel conditional computation, four reconfigurable comparison units and four reconfigurable computation units are used. **Figure 9** shows the block diagram of parallel comparison module, and **Fig. 10** shows the block diagram of parallel computation module. Conditional branch is widely used in the image processing to determine the following computation. Parallel comparison module can be configured by special instruction. The results of the comparison are stored in the condition registers for uncompleted comparison computation or for the following conditional computation. Parallel computation module can support SIMD computation such as 4 additions. Two ALUs in each unit can perform true branch computation and false branch computation at the same time. If the computation cannot be completed in one cycle, the results will be stored in temp registers and will be reused in next cycle.

Table 4 shows the special instructions which are added into PXL ASIP. Most of the special instructions are used to support 64-bit SIMD computation. LUT4 is used to execute the parallel 4 element look-up table access. LDPXL instruction can load data

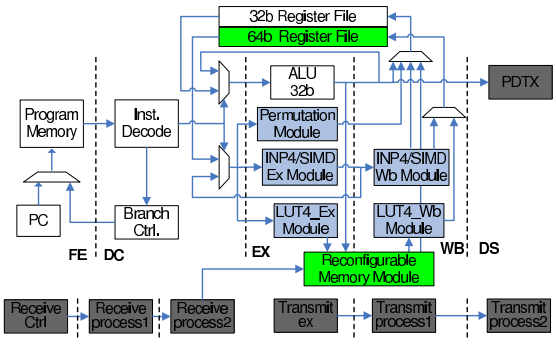

Fig. 7 Block diagram of PXL ASIP.

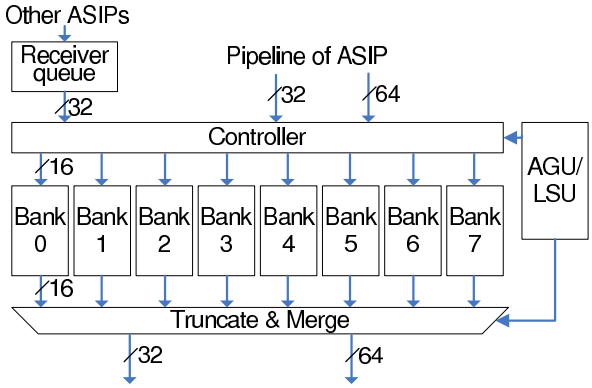

Fig. 8 Architecture of reconfigurable multi bank memory.

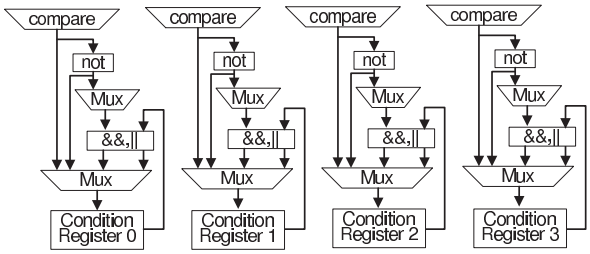

Fig. 9 Block diagram of parallel reconfigurable comparison module.

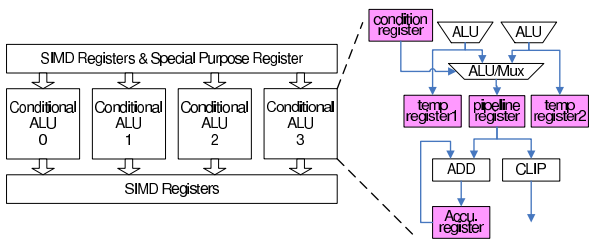

Fig. 10 Block diagram of parallel computation module.

Table 4 Special instructions in PXL ASIP.

Instruction	Function Description
INPR4	4 element inner product
LUT4	4 element look-up table access
CMPR4	4 element parallel comparison
COMP4	4 element reconfigurable computation
VR2GPR	Extract data or permute data from 64-bit register to 32-bit register
GPR2VR	Extract data or permute data from 32-bit register to 64-bit register
LDPXL	Load 4 channel pixel and increment access address
STM64	Store 64-bit data from a vector register to memory
LDM64	Load data from memory to a 64-bit vector register
ZOL	Zero overhead loop
SET_SPR	Initialize special registers (ex: coefficient registers, shift registers)

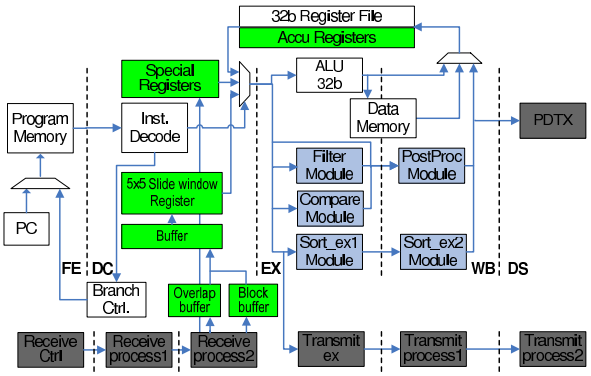

Fig. 11 Block diagram of 2D ASIP.

from 4 different banks and increase loading address simultaneously. It can effectively reduce the pixel data access time, and the loaded data can be immediately used by SIMD type instruction or special computing instruction such as COMP4 and INPR4 to improve the performance.

5.2 Architecture of 2D ASIP

2D ASIP is mainly designed for 2D filtering applications which are commonly used in pre/post processing such noise reduction, and edge enhancement. The limited range of the applications brings to the design of the ASIP with higher computational efficiency. **Figure 11** shows the block diagram of 2D ASIP. Additional function modules and registers which are shown in colored blocks are added into basic processor. Overlap buffers and block buffers are used to store pixel data from other ASIPs. The slide window module is used to improve the data access speed. The filter module is mainly responsible for the processing of 2D filtering. The compare module is used to handle the processing which has conditional decisions such as edge enhancement. The sort modules (Sort_ex1 and Sort_ex2) which can sort and determine maximum value, minimum value and median value is responsible for sorting the processing. The PostProc module is responsible for the post processing of the modules in EX stage. When processing in EX stage is unfinished, the rest of the processings are executed in PostProc module. The processed pixel can be transmitted to other ASIP by PDTX or to be stored in registers.

In our previous work [21], a 3×3 slide window register module

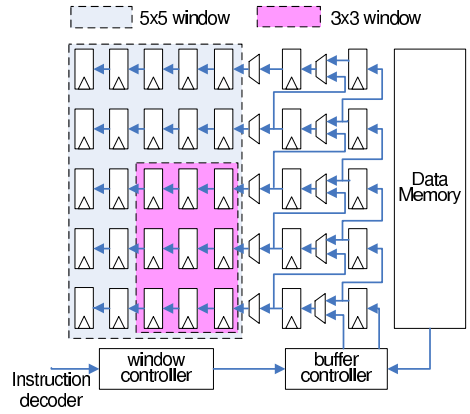

Fig. 12 Block diagram of slide window register module.

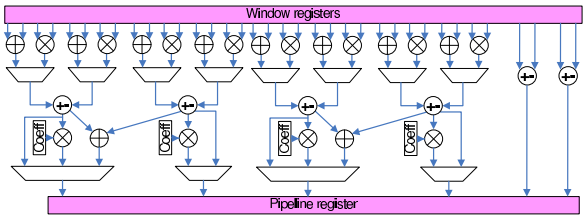

Fig. 13 Block diagram of filter module.

was designed to improve the accessing of pixel data from memory. In order to achieve higher performance, we modify this module so that it can automatically load data from memory during memory idle time. **Figure 12** shows the block diagram of the slide window module. After initialization, buffer controller starts to load data from memory and shift it up automatically. When window shift instruction is decoded by instruction decoder, window controller shifts window registers to the left. For 3×3 filtering applications, only the registers in bottom right corner are used to save shift time. When the size of the processing window is bigger than 5×5, the normal memory access instructions are used to load the uncover data.

Figure 13 shows the block diagram of a filter module. The design of the filter module has been introduced in Section 4. It is designed to improve the performance of the filtering operation such as multiply-after-sum or add-after-multiply. The computation of 5×5 filtering operation usually can be completed in one or two cycles.

The sorting module is implemented in two pipeline stages, the operation in the first pipeline stage is used to perform column sorting, and the operation in the second stage is used to perform row sorting. The sort instruction controls the type of sorting by selecting the needed output from sorting logics. The details of the sorting module is depicted in **Fig. 14**.

The reconfigurable datapath in post processing (PostProc) module which is mainly designed to complete the processing after EX pipeline stage is shown in **Fig. 15**. It can support 6 types of computation.

1. Add_Add_Mul_Clip. Four inputs which come from pipeline registers are summed up, and then it is multiplied by a coefficient, followed by the clip operation. Smooth filtering uses this operation.

2. Sum4. Four inputs which come from pipeline registers are

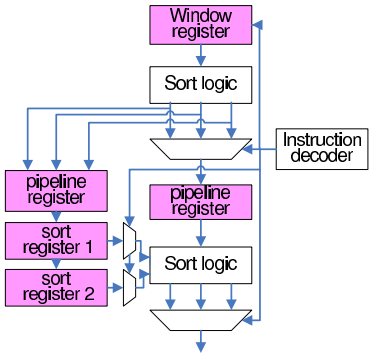

Fig. 14 Block diagram of sorting module.

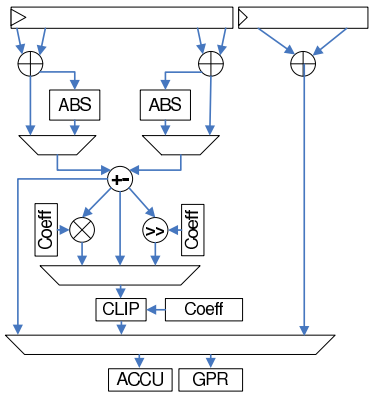

Fig. 15 Block diagram of post processing module.

Table 5 Special instructions in 2D ASIP.

Instruction	Function Description
SET_SPR	Initialize special registers (ex: coefficient registers, shift registers)
WINSH	Slide window loads new pixels or shift
ZOL	Zero overhead loop
FLT2D	2D filtering operation
MOV_SPR	Move data from window register to general purpose register
SORT	Provide maximum, minimum and median sorting
POST_PROC	Processing by using PostProc module
EE_POST	Post processing for condition-driven computation

summed up. It can be used to sum the result of 3×3 filtering in EX stage.

3. Sum4_Sum2. Four inputs and other two inputs which come from pipeline registers are summed up, respectively. It can be use to sum the result of 5×5 filtering in EX stage.

4. Add_Sub_Rsh. Two pairs of two inputs are added first, and the first result is subtracted from the second result. The edge-related processing can use this computation.

5. Add_Sub_Rsh_Clip. It has identical operations with Add_Sub_Rsh, with an additional clip operation added as the last operation.

6. Add_Abs_Add_Clip. It can be used to complete the processing which has magnitude computation such as edge detection.

The result after computation can be stored in the general purpose registers or the accumulation registers. PDTX transmission is activated in next pipeline stage, if the data is written into the assigned general purpose registers.

The details of the special instructions designed in 2D ASIP are explained in **Table 5**. Instruction SPR_SET is used to set the

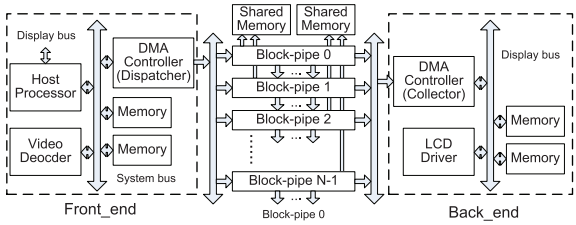

Fig. 16 Architecture of multi block-pipe image processing unit.

coefficients of multiplication, shift, clip and other initialization parameters, while WINSH is used to shift window register right. FLT2D controls the filter module and the PostProc module to perform the filtering processing. Sort module is activated by SORT instruction to execute the different types of sorting processing. POST_PROC controls PostProc module to finish the incomplete processing of the previous computation.

6. System Architecture

In order to support the processing of high resolution applications, we design a multi-ASIP parallel processing architecture. Since, the interconnection and communication between ASIPs affect the data flow, bandwidth and performance of image processing engine, the design of parallel architecture is focused on low communication overhead and high scalability.

6.1 Architecture of Image Processing Engine

Parallel architecture is designed based on the following assumptions. In pre-processing application, we assume that the data comes from camera sensor is connected to the system bus. In post-processing application, we assume that the data is decoded and stored in memory which is also connected to the system bus. The architecture of the processing engine which is shown in **Fig. 16** can be categorized into three parts; front-end, block-pipes and back-end, respectively. 1) Front-end includes one host processor, two memories, one DMA controller (dispatcher) and one video decoder. At first, host processor initializes the dispatcher. Then, the video decoder outputs the decoded block to memory. Once the data is gathered enough, the dispatcher moves a decoded block to a specific block-pipe according to its dispatch table. 2) A block-pipe consists of 3 kinds of ASIP. All types of image processing are performed inside the block-pipes, and several block-pipes are used to process an image in parallel to improve the processing performance. 1D_H ASIPs, 1D_V ASIPs and 2D ASIPs in a block-pipe are connected to the same type of ASIP in the next block-pipe and shared memories to process the boundary pixel processing. 3) Back-end operates in the opposite way to the front-end. In pre-processing application, the processed data is sent to encoder directly. While in post-processing application, pixel blocks are stored in memory and then sent to LCD driver in raster scan order. The number of block-pipes can be modified according to the performance requirement, but the maximum computing capability will be limited by the throughput of the front-end and the back-end.

6.2 Block-pipe Subsystem

Block-pipe is the basic processing unit in our image process-

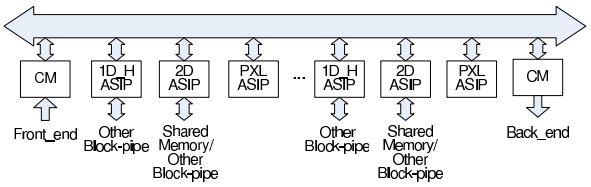

Fig. 17 Interconnection of block-pipe.

ing engine, and multiple block-pipe is used to process one image in parallel. **Figure 17**. shows the block diagram of a block-pipe. Three types of ASIP which are PXL ASIP, 1D ASIP and 2D ASIP together with two communication modules (CM) which are connected to the frond-end and the back-end are equipped inside. A full crossbar is used to connect these ASIPs and CMs. ASIPs which are used to execute the horizontal processing such as 1D_H ASIP (horizontal processing executed 1D ASIP) and 2D ASIP are connected to ASIPs in the next block-pipe to transmit necessary pixels. ASIPs which are used to execute vertical processing such as 1D_V ASIP (vertical processing executed 1D ASIP) and 2D ASIP can store pixels which are not able to be processed in current block into its data memory or shard memory. Moreover, the design of block-pipe has also been considered its scalability so that the users can modify the number of block-pipe depends on the performance requirement.

7. Experiment and Comparison

Processor Designer [22] is used to implement all ASIPs. This is aimed to shorten the development time of the ASIPs by describing the architecture in high level architecture description language (LISA) [23]. ASIPs in an image signal processing engine are designed based on the same basic processor to save the development time of basic instructions. The details of processor such as pipeline registers, stall mechanism, external memory interface can be handled by LISA, hence the designer can focus on the architecture design, and subsequently improve the development efficiency. Processor Designer is also used to generate cycle-accurate SystemC [24] model, HDL and SW development tools (assembler and linker). The architecture of the whole image processing engine is built at system level using the commercial ESL tool [25]. All cycle accurate simulations in this section are carried out on this platform.

7.1 Flexibility Evaluation

The ASIPs are designed to perform high performance in various applications. To estimate the flexibility of the proposed ASIPs, several applications are run on the proposed ASIP, GPP (ARM946), a DSP (TMS320C64x) and our previous work (TCTPE-2DX) in order to put their performance in comparison. Applications executed by ARM946 are compiled using ARM C Compiler (armcc) and evaluated using Realview debugger [27]. Applications executed by TMS320C64x are compiled using TI compiler (CCS) with level 3 optimization or TI image library [28] and they are then evaluated using DSP development board.

The design concept of PXL ASIP is to support a wider range of pixel level image processing with reconfigurable multi bank memory module and SIMD type computation modules. To prove the flexibility of PXL ASIP, three types of processing are tested. Color conversion is used to test the performance of inter channel computation. The performance of parallel memory access is tested by gamma correction. Dithering is used to test the performance of processing which includes conditional computation. The simulation results of different applications are shown in **Table 6**. In order to estimate the performance without the influence of process technology and circuit level optimization, the number of cycles per pixel is used as the unit. Color conversion can be completed within 4.2 cycle/pixel that include pixel loading, core processing and pixel storing. The address generate unit of the reconfigurable multi bank memory module automatically computes the pixel address and increase the address after loading by executing LDPXL instruction. LDPXL contributes 1 cycle/pixel loading. The core processing of color conversion is executed by using the parallel computation module for 3 cycles. In gamma correction, 4 channel data are loaded by using LDPXL instruction. LUT4 instruction performs 4 look-up table access operations for 1 cycle. Two VR2GPR instructions are used to move the data to the general purpose register and transmit to next ASIP. The 5.36 cycles/pixel of the computation of dithering mainly benefit from the parallel reconfigurable comparison module and the parallel computation module. COMP4_ALU0, CMPR4 and COMP4_COND instructions are massively used in dithering. By Comparing other processors, it can be concluded that the excellent performance is achieved by using SIMD type special instructions and reconfigurable multi bank memory module.

Table 6 Comparison of execution cycle per pixel of pixel level processing with other processors for different applications.

Processor	Color conversion	Gamma correction	Dithering
ARM946	71.3	15.9	25.0
TMS320C64x	29.8	6.4	35.8
TCTPE-2DX	8.9	11	41
PXL ASIP	4.2	3.3	5.36

ARM946: 441 MHz, TMS320C64x: 600 MHz, TCTPE-2DX: 500 MHz, PXL ASIP: 400 MHz.

Table 7 Comparison of execution cycle per pixel of 2D image processing with other processors for different applications.

Processor	Color Interpolation	3×3 Edge Detection	Median Filtering	5×5 Edge Detection
ARM946	68.6	81.5	218	147
TMS320C64x	12.3	2.61*	4*	5.8*
TCTPE-2DX	9.32	7.36	44	34.7
2D ASIP	5.06	3.54	3.66	5.9

ARM946: 441 MHz, TMS320C64x: 600 MHz, TCTPE-2DX: 500 MHz, 2D ASIP: 400 MHz. * TI IMGLIB.

Since 2D ASIP is designed focusing on 2D filtering, we test four typical 2D processing which are commonly used in the pre/post processing to examine the flexibility of the 2D ASIP. The results of the simulation are shown in **Table 7**. Color interpolation generates the lack channels of each pixel by referring to the neighboring pixels. The filter module and the post processing module of 2D ASIP are used to achieve 5.06 cycle/pixel. 3×3 edge detection is used to test the processing that the size of the filter as 3×3, as it is widely used in the high pass filter. The

Table 8 Improvement of PXL ASIP compared to our previous works.

Processor	Color Conversion		Gamma Correction		Dithering	
	CPP	Speedup	CPP	Speedup	CPP	Speedup
TCTPE	41.29	1×	14.08	1×	44.18	1×
TCTPE-2DX	8.92	3.15×	11	1.28×	41	1.07×
PXL ASIP	4.15	9.95×	3.27	4.3×	5.36	8.24×

CPP: Cycles/Pixel.
TCTPE: 400 MHz, TCTPE-2DX: 500 MHz, PXL ASIP: 400 MHz.

Table 10 Synthesis results comparison.

	ARM946E-S [26]	TMS320 C64X [29]	TCTPE (original CM) [19]	TCTPE-2DX [21]	2D ASIP	PXL ASIP
Technology	90 nm	130 nm	180 nm	90 nm	90 nm	90 nm
Frequency	441 MHz	600 MHz	100 MHz	500 MHz	400 MHz	400 MHz
Memory/Cache size (P$/D$)	8 kB/8 kB	16 kB/16 kB	4 kB/4 kB	4 kB/4 kB	4 kB/8 kB	4 kB/8 kB
Area (mm^2)	0.613 (w/o cache)	72 (with cache)	0.49 (w/o cache)	0.222 (w/o cache)	0.219 (w/o cache)	0.264 (w/o cache)
Normalized area (to 90nma)	0.613	34.5	0.123	0.222	0.219	0.264

slide window register module with 3×3 window mode, the filter module and the post processing module are used to provide 3.54 cycle/pixel in 3×3 edge detection. 5×5 edge detection is used to show 2D ASIP has no significant decrement in performance even in the larger filter size. In 5×5 edge detection, The change is only the mode of slide window register module. Median filtering is used to test the performance of sorting module, 3.66 cycle/pixel is achieved. Although, 2D ASIP cannot gain the better performance against DSP, the slower performance over DSP is acceptable in our case. Moreover, the proposed ASIPs have lower working frequency and extremely small area as compared to DSP, because lower power consumption and lower cost is more significant for embedded systems.

7.2 Performance and Area Evaluation

In order to support high resolution image processing, the target performance of ASIPs and the implementation conditions have to be defined. Our initial target of performance is to process Full HD image using an ASIP, with the working frequency of ASIPs at 400 MHz. The maximum allowable processing cycles for each pixel can be calculated by dividing the working frequency with the number of pixel for one second, that is $400\,\text{M}/(1{,}920 \times 1{,}080 \times 30) = 6.43$ cycles per pixel.

We compare the proposed ASIP with our previous works to show the performance improvement. TCTPE is the basic RISC processor which is used to design ASIPs. TCTPE-2DX ASIP [21] is the ASIP that we have developed to process 2D processing and pixel level processing. Its flexible datapath can support various applications, but its flexibility scarifies the computational efficiency. In order to satisfy the performance requirements, we decide to design two ASIPs for each dimension of image processing. We use the same applications which are used in previous subsection to estimate the improvement of performance. The performance of PXL ASIP for different pixel level processings improves from 4 times to 10 times against the basic RISC processor, and it shows 3 times to 8 times improvement against TCTPE-2DX. The SIMD type comparison instruction, reconfigurable arithmetic instruction and parallel memory access instruction provide high efficient computation for most of the pixel level image processings are the main reason of excessive improvement in PXL ASIP. The performance of new 2D ASIP for 2D different processing improves from 5 times to 40 times against the basic RISC processor. Compared to TCTPE-2DX [21], the new 2D ASIP shows 2 times to 14 times improvement. The simulation results are shown in **Tables 8** and **9**.

Table 9 Improvement of 2D ASIP compared to our previous works.

Processor	Color interpolation (bilinear)		3×3 Edge Detection (Sobel)	
	CPP	Speedup	CPP	Speedup
TCTPE	29.40	1×	52.34	1×
TCTPE-2DX	9.32	3.15×	7.36	7.11×
2D ASIP	5.06	5.81×	3.54	14.79×
Processor	Median Filtering		5×5 Edge Detection (Sobel)	
	CPP	Speedup	CPP	Speedup
TCTPE	160.56	1×	161	1×
TCTPE-2DX	44.03	3.65×	34.7	4.6×
2D ASIP	3.66	43.87×	5.9	27.3×

CPP: Cycles/Pixel.
TCTPE: 400 MHz, TCTPE-2DX: 500 MHz, 2D ASIP: 400 MHz.

The synthesis results are shown in **Table 10**. The ancestor of these two ASIPs is TCTPE [19] which has a general purpose communication module. Its area is about 0.123 mm^2. TCTPE-2DX [21] added some special instructions into TCTPE to support the pixel level image processing and the 2D image processing. The area of TCTPE-2DX increases to 0.222 mm^2. In order to support high resolution image processing, general purpose communication module is removed and message-passing fashion the communication pipeline which is optimized for parallel image processing is equipped into the system. 2D ASIP which is designed based on TCTPE with message-passing fashion communication pipeline has 1.78 times larger area than its base processor. The area of PXL ASIP is 0.264 mm^2 due to SIMD datapath and 64-bit register file. PXL ASIP has 2.14 times larger area than its base processor.

8. Conclusion and Discussion

In this paper, we discussed the design of an image processing

engine. We choose ASIP as our processing element, because its flexibility and computational efficiency meet the requirements of pre/post processing. Since our ASIPs are designed based on a basic RISC processor which has no communication ability, the message-passing fashion communication pipeline is employed in the first stage. Then, we analyze the target applications and design special instructions to improve the processing performance of ASIPs. A reconfigurable multi-bank memory module and SIMD type computation pipeline make a maximum of 9.95 times improvement compared to its basic processor. New 2D ASIP improves the low computational efficiency drawback of its previous work. An improved window slide module increases the data access speed through automatic loading. Filter module, PostProc module and other additional modules which are optimized for 2D image processing can effectively reduce the processing time. A maximum of 43.87 times performance improvement is achieved. The implementation time of ASIPs is shortened by using high level architecture description language. However, the analysis of applications and the design of reconfigurable datapath are still to be done manually. We derive several design rules by going through these processes. Therefore, we believe these processes can be performed automatically, and the design automation can further shorten the development time of ASIP.

References

[1] TMS320DSC21-A High-Performance, Programmable, Single Chip Digital Signal Processing Solution to Digital Still Cameras, Texas Instruments, white paper (2006).

[2] Alba Pinto C. and Beric, A.: HiveFlex-Video VSP1: Video Signal Processing Architecture for Video Coding and Post-Processing, *Proc. 8th IEEE International Symposium on Multimedia* (2006).

[3] Arakawa, S., Yamaguchi, Y., Akui, S., Fukuda, Y., et al.: A 512GOPS Fully-Programmable Digital Image Processor with full HD 1080p Processing Capabilities, *ISSCC 2008/SESSION 16/LOW-POWER DIGITAL/16.4* (2008).

[4] Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R. and Shippy, D.: Introduction to the Cell multiprocessor, *IBM Journal of Research and Development*, Vol.49, No.4/5, pp.589–604 (July 2005).

[5] Mori, T., Ueda, Y., Nonogaki N., et al.: A Power, Performance Scalable Eight-Cores Media, Processor for Mobile Multimedia Applications, *IEEE Trans. Solid-State Circuits*, Vol.44, No.11 (Nov. 2009).

[6] Xiao, H., Isshiki, T., Li, D. and Kunieda, H.: Optimized Communication and Synchronization for Embedded Multiprocessors Using ASIP Methodology, *IPSJ Trans. System LSI Design Methodlogy*, Vol.5 (Aug. 2012).

[7] Kistler, M., Perrone, M. and Petrini, F.: Cell Multiprocessor Communication Network: Built for Speed, *IEEE Micro*, Vol.26, No.3, pp.10–23 (July 2006).

[8] Tota, S.V., Casu, M.R., Roch, M.R., Rostagno, L. and Zamboni, M.: A hybrid shared-memory/message-passing multiprocessor NoC-based architecture, *DATE*, pp.45–50 (2010)

[9] Paulin, P.G., Pilkington, C., Langevin, M., Bensoudane, E., Lyonnard, D., Benny, O., et al.: Parallel programming models for a multiprocessor SoC platform applied to networking and multimedia, *IEEE Trans. Very Large Scale Integr. (VLSI) Systems*, Vol.14, No.7, pp.667–680 (July 2006).

[10] Chen, J.C. and Chien, S.-Y.: CRISP: Coarse-Grained Reconfigurable Image Stream Processor for Digital Still Cameras and Camcorders, *IEEE Trans. Circuits and Systems for Video Technology*, Vol.18, No.9, pp.1223–1236 (2008).

[11] Tanaka, T., Furuta, T., Nishida, H., Yoshioka, K. and Kiyohara, T.: A Pixel Level Parallel Processing Architecture for Multi-Standard Video Codec, *International Conference on Consumer Electronics*, pp.349–350 (2006).

[12] Soares, R., Azevedo, A. and Silva, I.S.: X4CP32: A coarse grain general purpose reconfigurable microprocessor, *Proc. Int. Parallel Distrib. Process. Symp.* (Apr. 2003).

[13] Tanigawa, K., Hironaka, T., Kojima, A. and Yoshida, N.: PARS architecture: A reconfigurable architecture with generalized execution model design and implementation of its prototype processor, *IEICE Trans. Inf. Syst.*, Vol.E86-D, No.5, pp.830–840 (May 2003).

[14] Primecell Inter-Processor Communications Module (PL320), available from ⟨http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0306b/index.html⟩ (accessed 2012-06-28).

[15] Primecell DMA Controller (PL080), available from ⟨http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0196g/index.html⟩ (accessed 2012-06-28).

[16] Lee, J.W., Park, J.W., Yang, M.H., Kang, S. and Choe, Y.: Efficient Algorithm and Architecture for Post-Processor in HDTV, *IEEE Trans. Consumer Electronics*, Vol.44, No.1 (Feb. 1998).

[17] Urfianto, M.Z., Isshiki, T., Khan, A.U., Li, D. and Kunieda, H.: Decomposition of Task-Level Concurrency on C Programs Applied to the Design of Multiprocessor SoC, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E91-A, No.7, pp.1748–1756 (2008).

[18] Seo, S., Dreslinski, R.G., Woh, M., Chakrabarti, C., Mahlke, S. and Mudge, T.: Diet SODA: A Power-Efficient Processor for Digital Cameras, *Low-Power Electronics and Design* (*ISLPED*) (2010).

[19] Urfianto, M.Z., Isshiki, T., Khan, A.U., Li, D. and Kunieda, H.: Multiprocessor SoC architecture with effcient communication infrastructure and advanced compiler support for easy application development, *IEICE Trans. Fundamentals*, Vol.E91-A, No.4, pp.1185–1196 (Apr. 2008).

[20] Liao, H.C., Asri, M., Isshiki, T., Li, D. and Kunieda, H.: A Reconfigurable High Performance ASIP Engine for Image Signal Processing, *19th Reconfigurable Architectures Workshop* (May 2012).

[21] Liao, H.C., Asri, M., Isshiki, T., Li, D. and Kunieda, H.: A High Level Design of Reconfigurable and High-Performance ASIP Engine for Image Signal Processing, *IEICE Trans. Fundamentals*, Vol.E95-A, No.12, pp.2373–2383 (Dec. 2012).

[22] Processor Designer, Synopsys Inc. (online), available from ⟨http://www.synopsys.com/Systems/BlockDesign/processorDev/Pages/default.aspx⟩ (accessed 2012-06-28).

[23] Schliebusch, O., Hoffmann, A., Nohl, A., Braun, G. and Meyr, H.: Architecture Implementation Using the Machine Description Language LISA, *Proc. ASP-DAC 2002*, pp.239–244 (2002).

[24] SystemC, IEEE Std 1666 2005.

[25] Platform Architect, Synopsys Inc. (online), available from ⟨http://www.synopsys.com/systems/architecturedesign/pages/platformarchitect.aspx⟩ (accessed 2012-06-28).

[26] ARM946 Performance (online), available from ⟨http://www.arm.com/products/processors/classic/arm9/arm946.php⟩ (accessed 2012-03-28).

[27] Realview (online), available from ⟨http://www.arm.com/products/tools/software-tools/rvds/arm-rv-debugger.php⟩ (accessed 2012-03-28).

[28] TMS320C64x Image/Video Processing Library Programmer's Reference (2003).

[29] Agarwala, S., Koeppen, P., Anderson, T., Hill, A., et al.: A 600 MHz VLIW DSP, *IEEE International Solid-State Circuits Conference*, Vol.2, pp.38–39 (2002).

Hsuan-Chun Liao received his B.E. and M.E. degrees in electrical engineering from Taiwan University of Science and Technology in 2004 and 2006, respectively. Currently he is a Ph.D. candidate in Department of Communication and Integrated Systems at Tokyo Institute of Technology. His interests include Video Codec design, MPSoC design and EDA tool design.

Mochamad Asri received his bachelor degree in Computer Engineering from Tokyo Institute of Technology in 2011. Currently he is working toward his master degree at Tokyo Institute of Technology. His interests include Processor Architecture, MPSoC design and Image Processor.

Tsuyoshi Isshiki received his B.E. and M.E. degrees in electrical and electronics engineering from Tokyo Institute of Technology in 1990 and 1992, respectively. He received his Ph.D. degree in computer engineering from University of California at Santa Cruz in 1996. He is currently an Associate Professor at Department of Communications and Integrated Systems in Tokyo Institute of Technology. His research interests include high-level design methodology for configurable systems, bitserial synthesis, FPGA architecture, image processing, computer graphics, and speech synthesis.

Dongju Li received her B.S. degree from LiaoNing University and M.E. degree from Harbin Institute of Technology, China, in 1984 and 1987, respectively. She worked as an IC design engineer in VLSI Design Laboratory of Northeast Micro-electronics Institute, Electronic Industry Burean, China, from 1987–1993. She received her Ph.D. degree from Tokyo Institute of Technology in 1998. She is currently a Research Associate at Department of Communications and Integrated Systems in Tokyo Institute of Technology. Her current research interests are multiprocessor architecture and VLSI Design.

Hiroaki Kunieda received his B.E., M.E. and D.E. degrees from Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He was Research Associate in 1978 and Associate Professor in 1985 at Tokyo Institute of Technology. He is currently a Professor at Department of Communications and Integrated Systems in Tokyo Institute of Technology. He has been engaged in researches on distributed circuits, switched capacitor circuits, IC circuit simulation, VLSI CAD, VLSI signal processing and VLSI design. His current research focuses on VLSI multimedia processing, design for system-on-chip, VLSI signal processing, VLSI architecture and VLSI CAD. He is a member of IEEE CAS & SP Society and IPSJ.