Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 5, Number 1, pages 20–25, January 2016

On the Development of a Hardware Library for Embedded Image Processing Systems Based on Reassemble Functional Blocks

Takanori Kurihara
Graduate School of CIS
Hosei University
Tokyo 184-8584 Japan
takanori.kurihara.3z@stu.hosei.ac.jp

Yamin Li
Faculty of Computer and Information Sciences
Hosei University
Tokyo 184-8584 Japan
yamin@hosei.ac.jp

***Abstract*—With rapidly increased demands for machine vision, many image processing circuits for embedded systems have been developed recently. However, the implementation and verification of new algorithm circuits require many tasks such as the interface design and the timing adjustment. This paper aims to simplify these tasks by implementing a hardware library for image processing. The library contains bus and device interfaces and basic image processing function modules. These modules have a simple interface and can be easily reassembled like building blocks. It means that it is possible to configure various algorithmic blocks by combining existing blocks. In the latter part of the paper, we show how to construct a sample system by using the hardware library and compare the system performance with the software implementation. The sample implements a corner detection system based on the Harris algorithm. The implemented system is 64.63 times faster than the software implementation with the OpenCV library running on an ARM CPU. In the future, we will expand and enhance our library to respond to various image processing algorithms.**

***Index Terms*—image processing; hardware library;**

I. Introduction

Embedded systems that use "eye" have been attracting a significant amount of research attention in recent decades. For example, in the automotive system, various algorithms have been proposed for enhancing the safety and making the automatic driving for practical use. However, when we integrate algorithms in real systems, it is necessary to devise on the implementation. This is due that the general image processing algorithms require large processing capabilities. In other words, it is difficulty in terms of cost to mount a general purpose processor with enough performance for image processing to embedded systems. For this reason, the image processing circuit dedicated to a single function for embedded systems is being studied.

Among them, Microsoft's Kinect [1] is an important device in embedded systems for computer vision. This product incorporates a dedicated processor for performing functions such as recognition of the skeleton. By using the processor, Kinect performs the complex recognition process without compressing the gaming platform resources. In addition to this case, various versions of the embedded hardware, such as a car driving assist system, are being actively studied [2], [3].

But of course, it is difficult to design this kind of hardware for developers or researchers whose specialty is in software and algorithm design. It needs the knowledge of hardware design and implementation. Furthermore, it is not enough only to implement the algorithmic core part. A real system actually requires timing and function control logic, DMA (direct memory access) controller, device controller, and so on. This complexity of the system implementation work obstructs the demonstration of new ideas of image processing. While the new algorithms in the image processing field are being proposed, it becomes more important to accelerate the execution of the algorithms with high-speed hardware circuits.

In this work, we aim to solve these obstacles of the system implementation. As one solution, we propose and provide a *hardware library* that contains the wrapper (bus interface) modules and common functional blocks of image processing. We design the function blocks so that the reassemble of them becomes an easy work and therefore it makes the implementation of the system easy based on block diagrams. Thus, developers will be able to focus on the algorithm/architecture's core logic by using this library. As a result, it becomes easier for developers to verify and implement the new system than the use of the existing development flow.

II. Background and Motivations

Reflecting the growing demand of image or video processing functions for many devices, vendors are offering products that pack many image processing functions in the form of intellectual property (IP) cores. For example, Altera's *Video and Image Processing (VIP) Suite* [4] includes various functions such as color space conversion, alpha blending mixer, and 2D filtering and scaling. Xilinx Inc. also sells similar product, such as *Video and Image Processing Pack (VIPP)* [5].

These suites or packages provide a variety of functions that contribute to the rapid prototyping and development of products. However, using these products is not suitable for academic research. These IP cores are implemented as a black box; it is impossible for users to expand or revise the functions. As an example, when using Altera VIP suite that contains 3×3, 5×5, and 7×7 filtering IP cores, it is impossible to reuse these

IPs in the case of implementing a 9×9 filtering function. In other words, a new module must be implemented completely that allows 9×9 filtering. Moreover in this case, it is also necessary to implement a compatible module that corresponds to each interface of other IP blocks for the cooperation with them.

In this paper, we aim to achieve the high flexibility of the image processing systems' construction. As a specific approach, we propose the hardware IP library based on the building block-like architecture. Block components of the library are able to be freely combined and we can construct the desired features by combining the blocks.

Furthermore, we adopt a simple interface to functional blocks for easily implementing the additional (user specific) blocks. To solve the shortage of functions of the library, it is only necessary to extend the existing blocks or implement a new block based on the simple interface.

III. Architecture of Hardware Library

A. Aim and Structure of the Library

The aim of the library is to provide the ease of development and reusability of the hardware components for embedded image processing systems. Ease of development achieves prototyping in a short period of time. Also, the existing components with high reusability allow the construction of various systems at low development cost. We designed the system architecture and interface of components based on these ideas.

In the field of image processing, algorithms are often represented by a block diagram with a simple data flow. Each block receives the image data and outputs the result of some operations. Our basic idea is based on the modularization of these "Blocks" which are designed in a hardware description language. We can construct desired systems by representing the block diagram using these Blocks. Such data flow and block diagram-like architecture makes it easy to understand for software engineers. In addition, we also considered that this construction can be done through a high-level synthesis based on the software description in the future.

In our library, as the most important thing, all image processing functions are constructed from primitives or combined functional Blocks. The "Filtering Block by kernel convolution" and "Color conversion Block" are two examples of such functions. These Blocks have a simple and unified interface so that they can be connected each other. When the complex features are needed, we can make up a new Block easily by combining the existing Blocks. Also it is useful for the implementation of a completely new or system-specific feature by combining the existing blocks partially. For these cases, the simple interface makes it easy to implement a new block.

Fig. 1 shows the architecture overview of the system adopted by our library. Image Processing Unit is the outside layer that connects to a system bus. Image Processing Blocks constitute the core part of the library.

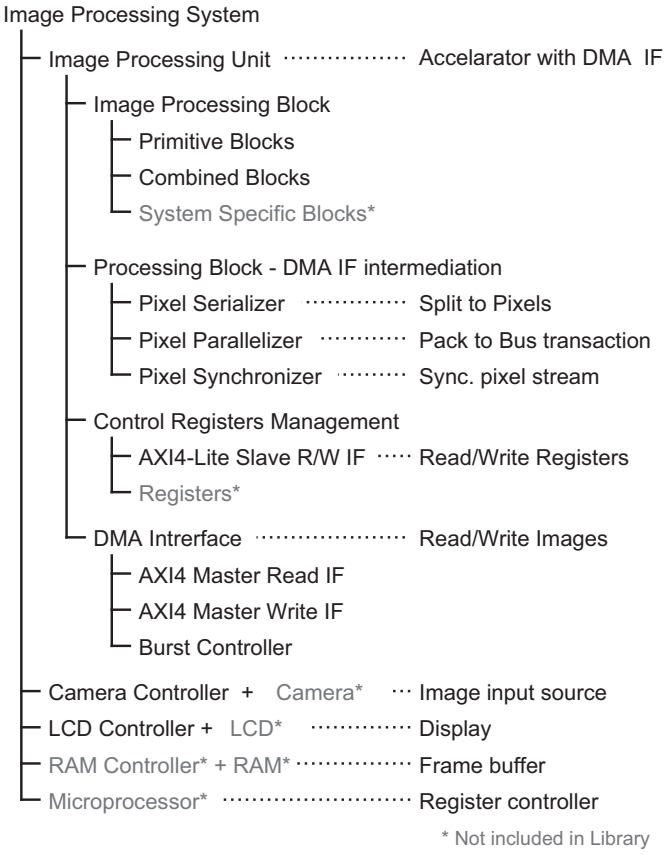


Fig. 1. Proposed architecture overview

B. System Architecture

Fig. 2 shows the block diagram of the system which will be implemented with our library. The system consists of a camera controller, an image processing unit, an LCD (liquid crystal display) controller, a microprocessor for register control (CPU), and memory (RAM) and its controller. The video stream will be processed in the following steps.

1) The Camera Controller stores the captured image into the memory.
2) The Image Processing Unit reads the captured image from memory, and writes the processed image back to the memory.
3) The LCD Controller reads the result image from the memory and outputs it to the LCD.

The Image Processing Unit consists of multiple modules which we will describe late. Except for the final image data, the temporary data will not be written to the memory; they are stored in the buffer inside the unit, in order to achieve the high processing speed of the unit.

The library provides a template design of Image Processing Unit to assist the implementation of the system. In most cases, it is easy to construct a system based on the template. In addition to the library of the Image Processing Unit, we prepared a camera controller and an LCD controller for the actual test. The CPU is used only for the register control of each peripheral in principle, and the memory is used as the

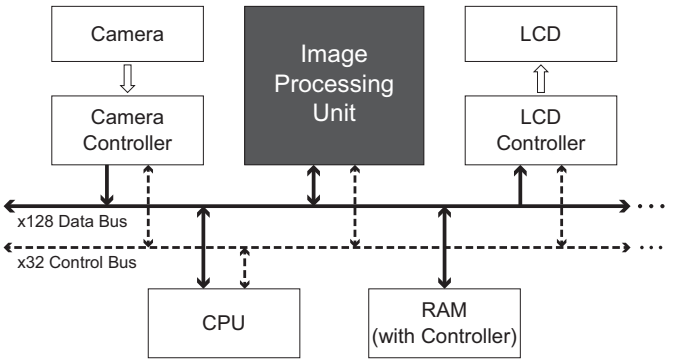


Fig. 2. Template architecture of the image processing system

input frame buffer from camera and the output frame buffer to the LCD controller.

In our system design, the AXI [6] bus standard is used. It supports the DMA operation. In the Image Processing Unit, the AXI bus interfaces and the DMA controller are provided. We implemented two types of interfaces of the AXI bus: the master interface for data transfer and the slave interface for register control.

The master AXI bus interface performs the memory data burst read and write. We designed it based on the AXI4 bus protocol. The information of the write destination or read source area is stored in the registers inside the Image Processing Unit. The slave AXI bus interface accepts register write or read access from bus master such as the CPU. We designed it based on the AXI4-Lite bus protocol.

The Image Processing Unit performs the operations based on the commands stored in the registers that are written by the program running on the CPU. Since the CPU never does other applications than the command transfer, a low-cost CPU can be used in our system.

The built-in camera controller supports the ITU-656 standard [7]. ITU-656 standard transfers the video data in the YUV422 format which is defined in the ITU-601 standard [8]. It is possible to immediately extract 8-bit brightness (Y) from YUV422 formatted pixel data. Because many image processing algorithms or systems use only the brightness information of image, this format is convenient for our system. If other formatted camera is used, it is necessary to develop a new controller that supports the standard. The LCD controller performs the video output based on the VGA standard.

C. Image Processing Unit Architecture

The hardware library is mainly used to construct the Image Processing Unit which connects to the AXI Bus. For the easy use of the library, we prepare a template design of the Image Processing Unit, as shown as in Fig. 3. Based on this template, developers can re-configure the Image Processing Unit by replacing registers and image processing blocks with the modules provided in the hardware library to meet their own requirements. The dark-gray blocks in the figure are the system-specific parts that are changeable.

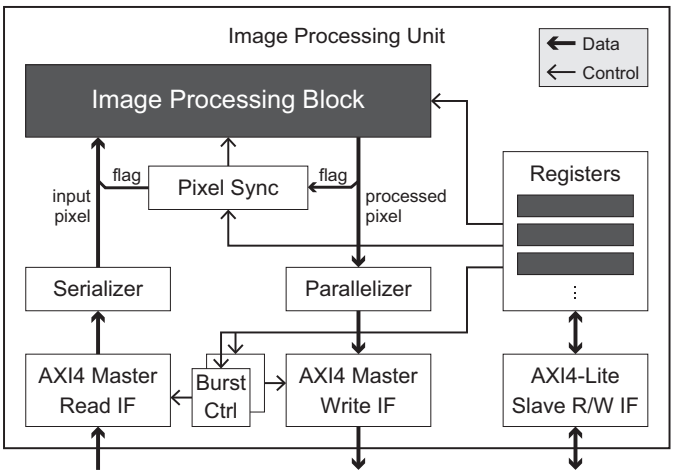


Fig. 3. Template block diagram of the image processing unit

The Image Processing Block is at the core of the Image Processing Unit. Besides providing the image processing functions, the other objective of the Image Processing Unit is to simplify the interface of the Image Processing Block and hence increase the productivity of the system. To achieve this objective, we provide the wrapper modules for the Image Processing Block and incorporate them to the Image Processing Unit. The wrappers mainly implement the following three functions: 1) DMA transfer of pixel data using the AXI Data Bus; 2) acceptance of register control using the AXI-Lite Control Bus; and 3) synchronization of the processed data based on the flag signal.

The details of the Image Processing Block will be described in the next subsection. The modules of AXI4 Master Read IF (interface) and AXI4 Master Write IF perform the DMA transfer of pixel data between the RAM and the Image Processing Block. The modules of Serializer and Parallelizer perform the conversion between parallel and serial data. The commands in the Registers module are written by the CPU via the AXI4-Lite Slave R/W (read/write) IF, and these commands are used to control the operations of the Image Processing Block and DMA controllers. The CPU can also read the state information from the registers. The Pixel Sync module performs the synchronization of the processed data based on the flag signal which will be described in the next subsection.

When you try to build a system using the library, if the data flow of your system matches the template, you can use the pixel data transfer mechanism of the template. For the small change, such as the modification of the number of bits per pixel, it is possible to cope with the new system by re-configuring the synthesis parameters. Even if you want to create a different flow, by adding a set of modules related to the pixel data transfer, it is possible to configure the system with multiple inputs and outputs, like the stereo matching required. However, in such a case, it is necessary to arbitrate the multiple data transfers appropriately inside the block without modifying the external interface. On the other hand, the register control mechanism can be remained without changing

from the template in most cases, and it is enough to configure your own system by adding or removing the system-specific registers.

D. Image Processing Block Architecture

The Image Processing Block implements various image processing functions, and a new block can be made up by combining multiple blocks. The hardware library provides some blocks of frequently used functions in various image processing algorithms. In this paper, we refer such a basic module as "primitive". Fig. 4 shows an interface of a typical primitive. The meaning of each port is listed in Table I. The descriptions of some implemented primitives will be given in the next section.

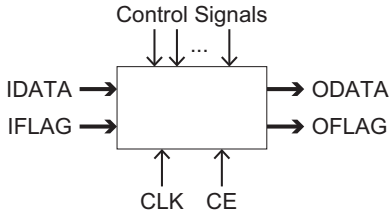


Fig. 4. Interface of a typical image processing primitive

TABLE I
PORT DESCRIPTIONS OF TYPICAL IMAGE PROCESSING PRIMITIVE

Port	Description
IDATA	Pixel data input. Data width configured by synthesis parameter.
ODATA	Pixel data output.
IFLAG	Pixel flag input.
OFLAG	Pixel flag output.
CLK	Pixel clock input. All image processing blocks are synchronized by the clock.
CE	Clock Enable input. If CE is low, all image processing blocks stall their pipelines.
Control Signals (Any port name)	Signals obtained from the registers or other blocks. They should be updated at idle or reset state.

The operations of the primitives are synchronized by the signals CLK (clock) and CE (clock enable). The operations of the primitives are determined by the control signals obtained from the registers, constant parameters, or other blocks. In principle, the input signals of a primitive are the pixel data and flag, and the output signals are the processed pixel data and delayed flag, corresponding to the latency of the pixel processing inside the primitive. The flag shows the pixel position in the image. It is used for the timing control of the frame processing and process switching in the border region.

Developers can incorporate any other features to the system by implementing additional blocks based on this interface. Also it is possible to build complicated processing blocks by combining multiple primitives and user defined blocks.

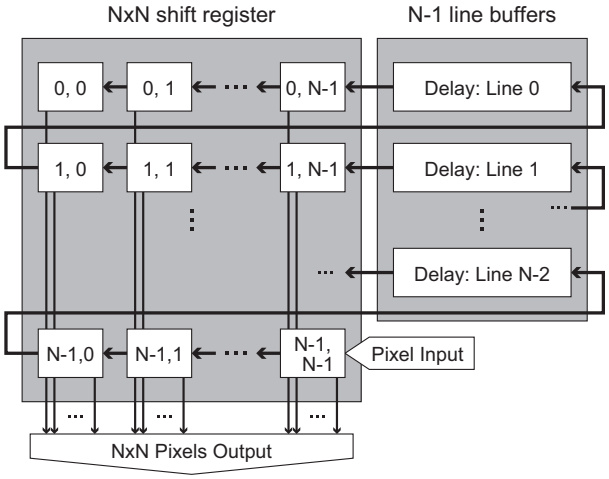


Fig. 5. Block diagram of the window buffer block

IV. Implementation of Primitive Blocks

In this section, we describe the implementation of some frequently used primitive blocks. These primitive blocks are fully parameterized, and it is possible to optimize the configuration for the target system.

A. Pixel Delay Block

Pixel Delay Block outputs the delayed data. The delay amount in clock cycles is configured by the register. This block is used as the synchronizer of pixel streams or the line buffer. It is constructed by a dual-port RAM (DPRAM) that can perform both the write and read operations independently at the same time. We implement the DPRAM with the first-in-first-out pixel buffering mechanism.

B. Window Buffer Block

Window Buffer Block provides the accesses to all the pixels in a window area simultaneously. Many image processing algorithms frequently require the information of the neighbor pixels. And in most cases, the necessary area is a square region that contains the focused pixel at the center. To achieve this behavior, the Window Buffer Block is constructed with Pixel Delay Blocks and shift registers as a line buffer as shown in Fig. 5. For example, a 5×5 window buffer can be generated using four Pixel Delay Blocks and five 5-stage shift registers.

C. Convolution / Filtering Block

Filtering is a general operation in the image processing. This operation needs the convolution as shown below.

$$C_{x,y} = \sum_{i,j} I_{x+i,y+j} \cdot K_{i,j}$$

where I, K, and C are the input image, kernel matrix, and result image, respectively.

The library provides this operation as the Convolution Block. And also, we provide the Filtering Block that is constructed with the Window Buffer and Convolution Block for performing any filter by only giving the kernel matrix.

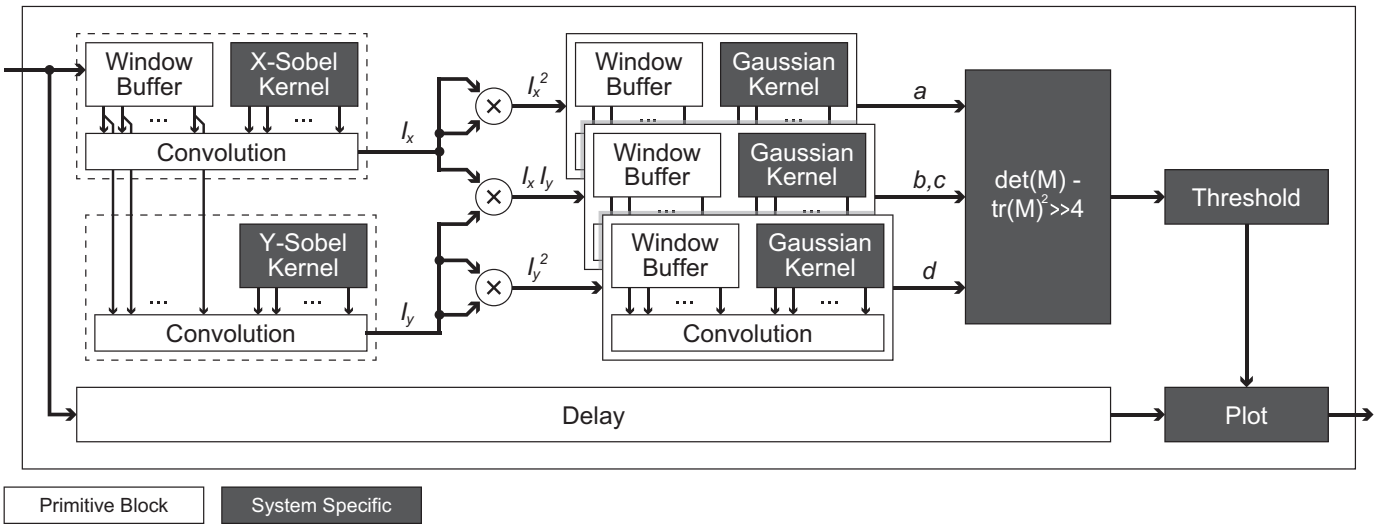


Fig. 6. Block diagram of the Harris corner detector block

V. Implementation of a Sample System

This section shows how to use the proposed hardware library to quickly build an image processing system. As an example, we implement a corner detection system based on the Harris corner detection algorithm [9]. This approach requires a calculation of the feature amount $R_{(x,y)}$ represented by the following equation.

$$\begin{aligned} R_{(x,y)} &= det(M_{(x,y)}) - k \cdot (tr(M_{(x,y)}))^2 \\ M &= \sum_{i,j} w(i,j) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \\ k &\approx 0.04 \sim 0.15 \end{aligned}$$

where I_x and I_y denote the x/y-direction partial differential image; $det(M)$ and $tr(M)$ means the determinant and the trace of matrix M, respectively. This can be done by the following translation.

- Partial differential image I_x, I_y can be represented by an X- and a Y-axis Sobel filters.
- Weighted sum $\sum_{i,j} w(i,j)[\cdots]$ can be represented by a Gaussian Filter.

With these replacements, the matrix $M_{(x,y)}$ can be derived in a sequence of very common blocks: Sobel filter, multiplier, and Gaussian filter. These filters can be easily configured by using the Filtering block. We use one Window Buffer Block and two Convolution Blocks for Sobel filters because these two filters share the same window. After that, we need to implement only the circuit for calculating the determinant and trace of $M_{(x,y)}$ and final feature quantity $R_{(x,y)}$. In this case, for the 2×2 matrix M, determinant and trace will be calculated by the following equations.

$$\begin{aligned} det \begin{pmatrix} a & b \\ c & d \end{pmatrix} &= ad - bc \\ tr \begin{pmatrix} a & b \\ c & d \end{pmatrix} &= a + d \end{aligned}$$

The implementation of these equations is not difficult because it can be calculated by the additions and multiplications. In addition, we set $k = 2^{-4} = 0.0625$ in the $R_{(x,y)}$ calculation. Thus the multiplying by k can be replaced by shifting the data to the right by four bits.

We implement the system so that the detected corners are marked on the original image. To achieve this effect, the post processing and the original image delaying are required. Fig. 6 shows the block diagram of the Harris corner detector block. The Threshold block controls the display of the corner marks to the original image. Through implementing the top-level and gray colored blocks in the figure, the corner detector block was constructed. As demonstrated by X- and Y-axis Sobel filters, the partial block sharing is also an advantage of this library.

VI. Experimental Results

In this section, we evaluate the performance of the system implemented in the previous section. To get the speedup, we compare the system performance of the hardware-implemented system with that of the software-implemented system. For the evaluation of the actual implementation, we use an FPGA (field programmable gate array) evaluation board which contains a hardwired ARM CPU. As already mentioned in the section III-B, the CPU's calculation power is not required for the hardware-implemented system because it performs only the register control. However, in order to experiment under the same conditions as much as possible (excepting the calculation circuit), we use the same device for these experiments. We use an Arrow SoCKit Development Board that mounts the Altera Cyclone V SoC 5CSXFC6D6F31C6N device and 1 GB DDR3 SDRAM with hard wired SDRAM controller. Also, we use the MTV-54K0DN CCD camera for capturing the image.

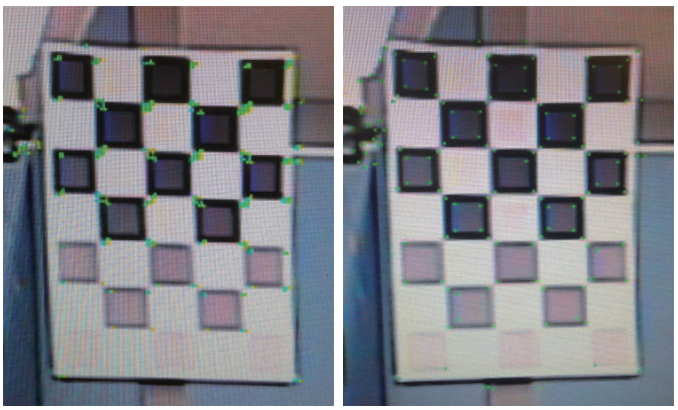
We use the Altera Quartus II 15.0.0 software for configuring the system on FPGA device. The programmable logic capacity of the FPGA and the utilization of the logic elements are shown in Table II. In addition, the distribution of the used logic elements is shown in Table III. The ALM (Adaptive

TABLE II
LOGIC CAPACITY AND UTILIZATION OF CORNER DETECTION SYSTEM

	Used	Available	Utilization
ALMs	2,694	41,910	6%
Block memory bits	721,572	5,662,720	13%
DSP blocks	57	112	51%

TABLE III
LOGIC DISTRIBUTION OF THE CORNER DETECTION SYSTEM

	Used	Ratio
ALMs		
Corner Detector	785.8	29%
LCD Controller	406.0	15%
Camera Controller	338.8	13%
Interconnect + Port	1,163.2	43%
Block memory bits		
Corner Detector	704548	98%
LCD Controller	8576	1%
Camera Controller	8448	1%
Interconnect + Port	0	0%
DSP blocks		
Corner Detector	49	86%
LCD Controller	7	12%
Camera Controller	1	2%
Interconnect + Port	0	0%



(a) Hardware implementation (b) Software implementation

Fig. 7. Result images of the corner detection system

Logic Module) in the table is a unit of the logic resource in Altera's FPGA device families. The VGA screen image during the execution is shown in Fig. 7(a). The corners are indicated by the green markers.

As described before, the software for the comparison runs on the ARM CPU. The software is developed under the Linux environment and implemented by using OpenCV 2.4.8 and mainly the cv::goodFeaturesToTrack() function. We configured each filter window with the size of 3×3, which is the same as in the hardware-implemented system. The VGA screen image during execution is shown in Fig. 7(b).

TABLE IV
PERFORMANCE OF THE CORNER DETECTION SYSTEM (3,000 FRAMES)

	Processing Time (sec.)	Average FPS
Software implemented	1189.21	2.52
Hardware implemented	18.40	163.03

The processing times for dealing with 3,000 frames are shown in Table IV. The hardware implementation achieves 64.63 times faster performance in term of FPS (frames per second) than the software implementation. The 163.03 FPS is good enough for the real-time processing.

VII. CONCLUSION

In this paper, we proposed an architecture of a hardware library that facilitates construction of image processing systems by reassembling block modules. The library provides various basic functions of image processing as primitive blocks. It is possible to realize various algorithms by flexibly combining these blocks. In order to achieve the faster and easier implementation of the image processing systems, we have prepared various useful primitive blocks which have a simple interface for connecting each other. We have constructed a real image processing system for the purposes of demonstrating the use of the library and evaluating the performance compared with that of the software implementation which runs on a general-purpose processor.

In the future, we will develop more image processing blocks and add them to the hardware library so that it will be able to adapt to various kinds of image processing systems. And also we will revise the architecture so that it has a more user-friendly interface for the configuration of the systems. This project is currently under development. We will make the hardware library public-accessible via Internet.

REFERENCES

[1] Z. Zhang, "Microsoft kinect sensor and its effect," *MultiMedia, IEEE*, vol. 19, no. 2, pp. 4–10, 2012.

[2] D. Mandal, J. Sankaran, A. Gupta, K. Castille, S. Gondkar, S. Kamath, P. Sundar, and A. Phipps, "An embedded vision engine (eve) for automotive vision processing," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, June 2014, pp. 49–52.

[3] C. Farabet, C. Poulet, and Y. LeCun, "An fpga-based stream processor for embedded real-time vision with convolutional networks," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, Sept 2009, pp. 878–885.

[4] (2015) Video and image processing suite megacore functions. Altera Corporation. [Online]. Available: https://www.altera.com/products/intellectual-property/ip/dsp/m-alt-vipsuite.html

[5] (2015) Video and image processing pack. Xilinx Inc. [Online]. Available: http://www.xilinx.com/products/intellectual-property/ef-di-vid-img-ip-pack.html

[6] ARM, "Amba 4 axi and ace protocol specification," 2013. [Online]. Available: http://www.arm.com/products/system-ip/amba-specifications.php

[7] "Recommendation itu-r bt.656-5," 2007. [Online]. Available: http://www.itu.int/rec/R-REC-BT.656/e

[8] "Recommendation itu-r bt.601-7," 2011. [Online]. Available: http://www.itu.int/rec/R-REC-BT.601/e

[9] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151.