IEICE TRANS. INF. & SYST., VOL.E101–D, NO.2 FEBRUARY 2018

PAPER Special Section on Reconfigurable Systems

A Describing Method of an Image Processing Software in C for a High-Level Synthesis Considering a Function Chaining

Akira YAMAWAKI[†a)] ***and*** **Seiichi SERIKAWA**[†], ***Members***

SUMMARY This paper shows a describing method of an image processing software in C for high-level synthesis (HLS) technology considering function chaining to realize an efficient hardware. A sophisticated image processing would be built on the sequence of several primitives represented as sub-functions like the gray scaling, filtering, binarization, thinning, and so on. Conventionally, generic describing methods for each sub-function so that HLS technology can generate an efficient hardware module have been shown. However, few studies have focused on a systematic describing method of the single top function consisting of the sub-functions chained. According to the proposed method, any number of sub-functions can be chained, maintaining the pipeline structure. Thus, the image processing can achieve the near ideal performance of 1 pixel per clock even when the processing chain is long. In addition, implicitly, the deadlock due to the mismatch of the number of pushes and pops on the FIFO connecting the functions is eliminated and the interpolation of the border pixels is done. The case study on a canny edge detection including the chain of some sub-functions demonstrates that our proposal can easily realize the expected hardware mentioned above. The experimental results on ZYNQ FPGA show that our proposal can be converted to the pipelined hardware with moderate size and achieve the performance gain of more than 70 times compared to the software execution. Moreover, the reconstructed C software program following our proposed method shows the small performance degradation of 8% compared with the pure C software through a comparative evaluation preformed on the Cortex A9 embedded processor in ZYNQ FPGA. This fact indicates that a unified image processing library using HLS software which can be executed on CPU or hardware module for HW/SW co-design can be established by using our proposed describing method.

key words: *high-level synthesis, FPGA, describing method, image processing, function chaining*

1. Introduction

The embedded products tend to include sophisticated image processing in order to make their own added value increase. In addition, many of them are desired to run long period by battery. That is, the embedded products must achieve the high performance and the low power consumption. To overcome such problem, the hardware implementation of the image processing is important. However, the hardware design could be a significant burden to the product developers. To reduce the designing load of the hardware, the high-level synthesis (HLS) technologies automatically converting the software to the hardware module in the hardware description language (HDL) have been researched and developed [1], [2]. From the viewpoint of device, the FPGA which has design flexibility due to the reconfigurable nature is attractive to overcome the short life cycle of the embedded products, compared with the ASIC.

The HLS has a good affinity to FPGA since the FPGA has the similar flexibility as the software. Thus, the HLS technologies on FPGA have been received a lot of attention and the well-established commercial tools have come to be provided [1], [2]. The HLS tools attempt to lead many software engineers to the hardware development on FPGA in order to support the embedded hardware engineering that a few skillful hardware engineers have proceeded. As a result, the FPGA also may become a main stream of the hardware implementation in the embedded systems. However, the pure software code cannot be converted to the highly optimized hardware module without careful considerations yet [1], [2]. The software code targeted to the HLS must be written taking into the account of the hardware specific structures like the pipeline, parallelism, memory port limitation, and memory buffering. This is one of the problems that disturb the software engineers from utilizing a HLS tool as the general purpose programming environment.

The providers and researchers of the HLS tool have shown describing methods of HLS software to generate an efficient image processing hardware [3]–[7]. They have concentrated on the individual single image processing primitives such as the gray scaling, color-space conversions, binarization, filters, DCT and so on. In general, the practical image processing have several primitives mentioned above as sub-functions. For example, the canny edge detection is built on the sequence of sub-functions that are the gray scaling, gaussian smooth filter, sobel filter, non-maximum suppression, hysteresis thresholding, and the edge thinning [8].

Some researchers have taken into account of connecting sub-functions and proposed the new languages for the image processing to be converted by their own HLS tools [9], [10]. These proposals establish the function connections by specifying the dedicated line buffers. However, these specific languages cannot be compiled by a generic compiler like gcc which supports many kinds of embedded processors. The embedded engineers first describe the system behavior in a general purpose language like C, C++, and Java, then explore the computation centric parts to be implemented as hardware [11]. Due to the performance profiling and the resource limitation of the hardware, the designer may decide to implement some part of image processing as the software. Performing such trade-off and hardware implementation in a unified general purpose language

Manuscript received March 30, 2017.
Manuscript revised August 10, 2017.
Manuscript publicized November 17, 2017.

[†]The authors are with the Faculty of Engineering, Kyushu Institute of Technology, Kitakyushu-shi, 804–8550 Japan.
a) E-mail: yama@ecs.kyutech.ac.jp

DOI: 10.1587/transinf.2017RCP0001

can make the HW/SW co-design process non-error-prone, without limitation of kind of the embedded processors to be used.

The Vivado HLS from Xilinx supporting C and C++ language [12] is seemed to be a most spreading HLS tool due to easy and free use. The manual of this tool shows a describing method of the main image processing function consisting of the sequential sub-functions which are provided as the specific image processing library [12]. However, this manual shows only an arrangement of sub-functions as APIs. Since the insides of the APIs are not opened, all engineers cannot apply this library and hidden describing methodology to their own image processing software.

This paper shows a systematic describing method of the C behavior including the chain of basic sub-functions to realize the image processing for a HLS tool in C language. To the knowledge of authors, few studies have focused on this aspect. The C language is most popular language for the embedded engineers [13] and HLS tools [1], [2]. Thus, our proposal can significantly contribute to spread the usage of the FPGA and HLS technology to the embedded software engineers who are thinking to use an FPGA.

According to the proposed method, any number of sub-functions can be chained by inferred FIFOs maintaining the pipeline structure across the whole of hardware. The image processing even if it has long function chaining can easily achieve the near ideal performance of 1 pixel per clock. For FIFO, the deadlock due to the mismatch of the number of pushes and pops on the FIFO connecting the functions becomes a problem. The proposed method eliminates it implicitly. In addition, the interpolation to the undefined border pixels caused by a window processing like special filter is automatically performed.

The rest of the paper is organized as follows. Section 2 explains the research motivation, indicating the research prerequisites about the C hardware description where this paper deals with. Then the research problems are shown caused by the conventional methods in those prerequisites. Section 3 shows the proposed describing method of hardware behavior in C to overcome the research problems. Section 4 develops a hardware platform to perform the practical experiments and demonstrates the case study applying our proposal to a canny edge detection. Then the results of FPGA implementation and performance evaluation are shown and discussed. Finally, Sect. 5 concludes the paper indicating future work.

2. Motivation and Problem

The HLS technology is good at converting the software showing the stream memory access patterns to the efficient hardware. Since the HLS is hard to handle the dynamic and random complicated memory access patterns, the hardware/software co-design combining the software execution doing the sophisticated memory accesses and the hardware execution doing the high performance stream processing are generally performed [7], [11]. To realize the pipelined hard-

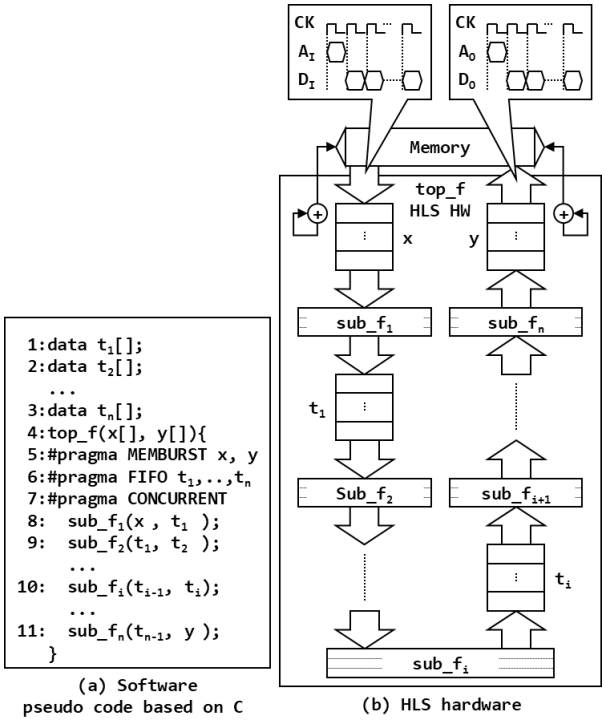


Fig. 1 Top function organization.

ware with the performance of 1 pixel per clock including the chained sub-functions shown in Fig. 1, each sub-function must be converted to the pipelined sub-hardware with 1 pixel per clock. In addition, the memory interface has to support a burst bus transfer.

This section summarizes the top function and sub-functions showing streamable memory accesses conventionally targeted by HLS. Then, we would like to indicate the research problems that appear when connecting the sub-functions on this conventional hardware description.

2.1 Top Function

Figure 1 shows an organization of the top function including a sequence of sub-functions. The top function as shown in Fig. 1 (a) includes the some sub-functions (sub_f_1 to sub_f_n). The sub-functions are connected by the arrays (t_1 to t_n).

For the arguments of the top function, the array of x is the input data and that of y is the output data. The arguments of the top function to the data array are generally assigned to the external memory by a HLS technology used. Although omitted, the top function is able to have some arguments for any parameters. Those parameters are generally assigned into the direct ports or the memory mapped registers. For the data arrays in the arguments, the HLS tools have some pragma to specify the memory ports as the bus interface supporting a burst transfer. In this figure, this pragma is indicated as `#pragma MEMBURST x, y` at line 5.

For the arrays connecting sub-functions, the HLS tools mostly try to assign them to embedded memories on an FPGA used as default. However, it is not practical that

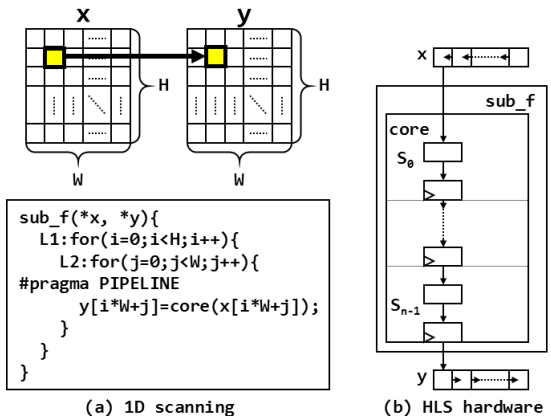


Fig. 2 1D scanning.

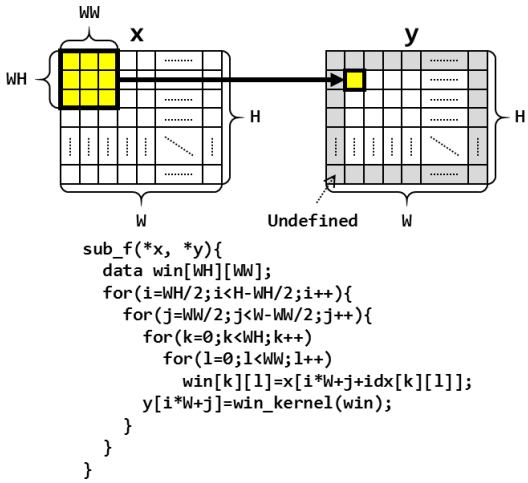


Fig. 3 Window processing from software view.

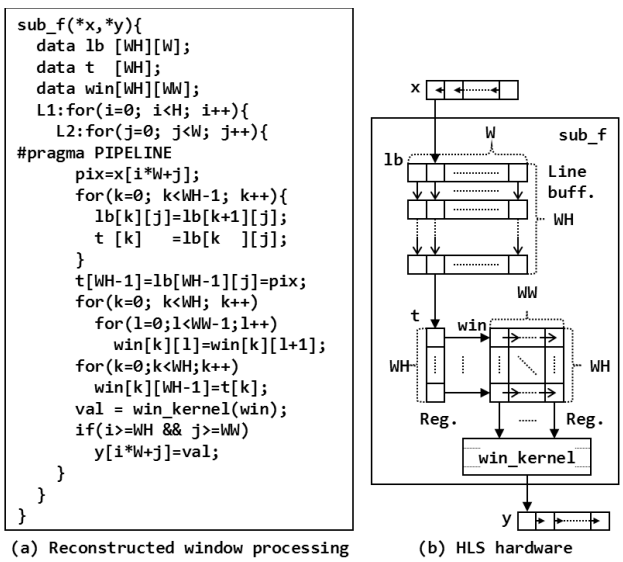


Fig. 4 Reconstructed window processing for HLS.

the whole image is stored to a small embedded memory whose capacity is limited. Thus, the HLS tool generally supports the FIFO as the communication media between sub-functions. The sub-functions across the FIFO have to act as a producer and consumer individually running. The producer pushes the processed data into the FIFO while the consumer pops them from the FIFO immediately. The FIFO have only to hold the needed data at communicating point not whole image. The HLS tools have pragmas to make the sub-functions concurrently run and to assign arrays to FIFOs bridging the sub-functions. These pragmas are described as `#pragma FIFO t1,..,tn` at line 6 and `#pragma CONCURRENT` at line 7.

The hardware generated by a HLS tool would have the structure shown in Fig. 1 (b). On this structure, the pipeline from the input port to the output port can be shifted every clock cycle unless a pipeline stall occurs due to the memory stall. As a result, the processing of 1 pixel per clock is achieved after some latency until the pipeline is filled.

2.2 Sub-Function: 1D Scanning

A memory access pattern of the 1D scanning accesses the single pixels successively in the raster scan order. The typical examples are *Thresholding* and *Color space conversions* [14], [15].

Figure 2 shows a pseudo C code and HLS hardware of the sub-function with the 1D scanning. As shown in Fig. 2 (a), the 1D scanning gets the single pixel and processes it in a core calculation. The processed single pixel is stored into the memory. The pixels previously loaded or processed can be used in the core calculation to make new pixel. Note that single read and single write are performed to respective arguments of the sub-function. In this case they are x and y which are converted to single read and write ports. If each argument has multiple accesses to different addresses, the HLS cannot make the pipelined hardware with 1 pixel per clock due to the port conflict.

By using a pragma for pipelining such as `#pragma PIPELINE` in Fig. 2 (a), the HLS generates a pipelined hardware such as Fig. 2 (b) handling 1 pixel per clock, while pipelining the core calculation.

2.3 Sub-Function: 2D Scanning

Figure 3 shows a window processing using the center pixel and its local neighbors as the typical concept of 2D scanning access pattern from the viewpoint of software. Since this type of the processes accesses the multiple pixels with different addresses from the arguments, the HLS cannot generate the pipelined hardware handling 1 pixel per clock due to the resource conflicts. The reconstruction methods of such software description have been proposed so that the HLS can generate an ideal pipelined hardware with 1 pixel per clock [3]–[5]. Figure 4 shows a pseudo C code of the reconstructed window processing and the HLS hardware respectively.

The line buffers (`lb`) are used to hold the image lines spatially accessed by 2D window. The line buffer has the same width as the image. The number of line buffers is

equal to the height of the window. Since the line buffer would have many entries to hold the line of the image, it would be assigned to the embedded memory of the FPGA used. The single input pixel is pushed from the input port into the targeted column of the line buffer. A temporal register file (`t`) is prepared to hold the targeted column from the line buffer. The 2D window is represented as a register array (`win`) whose row forms the shift register. The window kernel (`win_kernel`) performs a kernel processing using the values of the center and its neighbor pixels in this register array. The window processed pixel is output to the external port one by one. The actions mentioned above proceed clock by clock in a pipeline fashion. By limiting memory reading and writing to one, HLS technology can generate the ideal pipelined hardware with 1 pixel per clock without resource conflicts for the load/store.

2.4 Research Problem

The describing method of 2D window processing mentioned above is well-established for a single window processing. However, since chaining such window processing is not considered, some problems occur to generate the pipelined hardware built on the sequence of the reconstructed window processing.

For example, as shown in Fig. 5, a deadlock can occur on a FIFO connecting the window processing since the number of pushes and pops is differ. This example shows the connection of a smoothing filter using 5x5 window for the denoising and the sobel filter using 3x3 window to detect the edges. In the window processing, all pixels of the input image have to be traversed while the processed center pixel is stored into the output image. Thus, a window processing reads $W \times H$ pixels and writes $\{W-(WW-1)\} \times \{H-(WH-1)\}$ pixels. The W and H are the width and height of the image. The WW and WH are the width and height of the window. In this case, the smooth filter pushes the $(W-4) \times (H-4)$ pixels to the FIFO while the sobel filter tries to pop $W \times H$ pixels. The sobel filter cannot pop the rest of $4(W+H+4)$ pixels then the deadlock occurs.

In addition, concerned on generating the optimize hardware for a single window processing, this method applies a simplest way neglecting the undefined border pixels shown in Fig. 3. Leaving the border pixels undefined may badly affect the result of the window processing following.

So, this paper proposes a systematic describing method of C software on the chain of sub-functions shown in Fig. 2 and Fig. 4 to realize a relatively large image processing for a HLS tool. The proposed method maintains the pipeline structure across the whole of hardware while eliminating the deadlock across the FIFO and interpolating the border pixels.

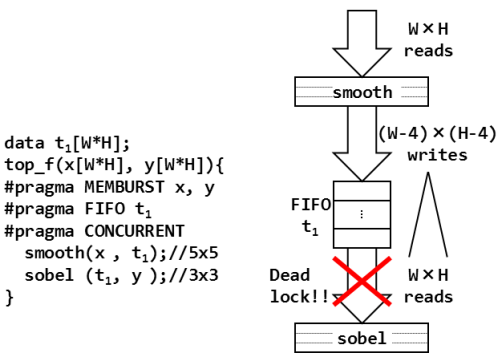


Fig. 5 Deadlock across FIFO.

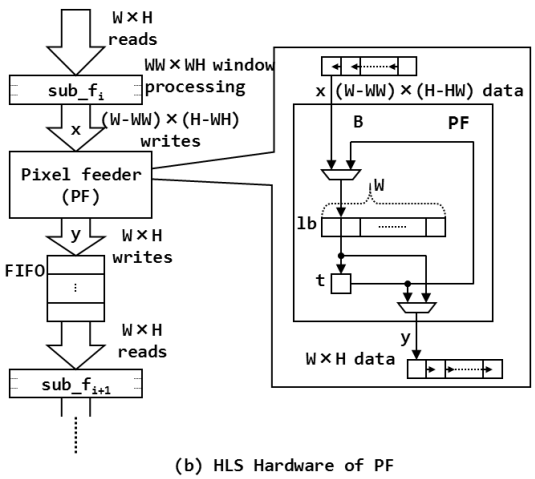
3. Proposed Method

Figure 6 shows an overview of the proposed C behavior and its HLS hardware. The key feature is the pixel feeder (PF) inserted after a sub-function performing a window processing. The PF interpolates the input pixels decreased by a window processing with the nearest neighbor pixels, making the number of pixels the same as the original image.

```
data t1[], t2[], t3[];
top_f(x[], y[]){
#pragma MEMBURST x, y
#pragma FIFO t1,t2,t3
#pragma CONCURRENT
  smooth(x , t1, W, H      );//5x5
  PF    (t1, t2, W, H, 5, 5);
  sobel (t2, t3, W, H      );//3x3
  PF    (t3, y , W, H, 3, 3);
}

PF(*x, *y, w, h, ww, wh){
  data lb[W], t;
  for( i = 0; i < h; i++ ){
    for( j = 0; j < w; j++ ){
#pragma PIPELINE
      if((i>=1&&i<=wh/2)||(i>=h-wh/2&&i<=h-1))
        *y++=lb[j];
      else
        if((j>=1&&j<=ww/2)||(j>=w-ww/2&&j<=w-1)){
          *y++=t; lb[j]=t;
        }else{
          lb[j]=*x++; *y++=lb[j]; t=lb[j];
        }
    }
  }
}
```

(a) Pseudo C Behavior



(b) HLS Hardware of PF

Fig. 6 Proposed C behavior.

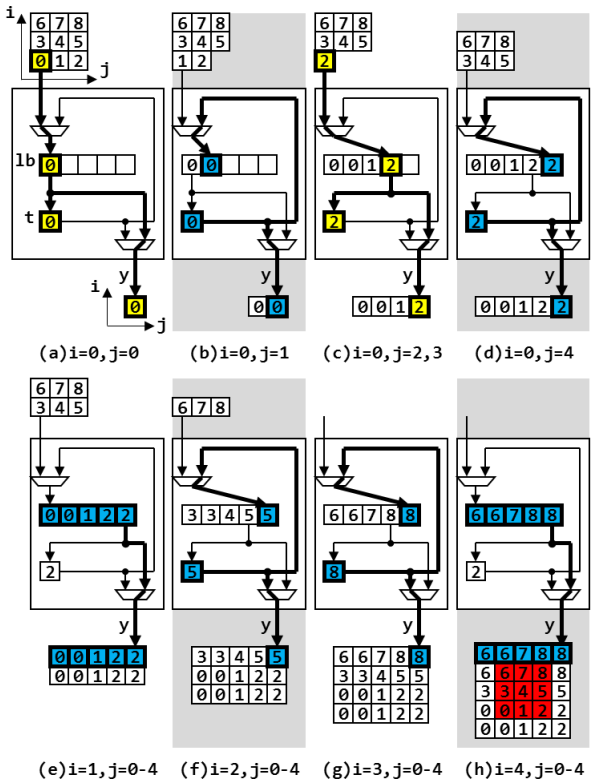


Fig. 7 Execution snapshots of pixel feeder.

The PF has the line buffer with the same width as the original image (`lb`) and the temporal register (`t`). For the border pixels on the left and right most side, the PF interpolates by duplicating the successive pixels to the line buffer and memory. For the top and bottom of the image, the content of the line buffer is replicated.

Figure 7 shows an execution snapshots of the PF. This example assumes that the pixel feeder interpolates input pixels decreased by a 3×3 window processing and the original image has 5×5 size. As shown from Fig. 7 (a) to (d), the PF stores the input pixels into the line buffer and memory, duplicating the border pixels with adjacent pixels. This line stored into the memory corresponds to the interpolated border line of the top of the processed image. As shown in Fig. 7 (e), the PF stores the content of the line buffer into the memory as the processed top line of the image. Then, the PF executes as well as Fig. 7 (a) to (d) for the middle part of image, interpolating the edge pixels on the lines as shown in Fig. 7 (f) and (g). To the bottom line of the output image, the PF stores the content of the line buffer as shown in Fig. 7 (h). As a result, the PF finishes pushing the pixels interpolated by the nearest neighbors with the same number of all pixels in the original image.

4. Experiment and Discussion

4.1 Experimental Setup

The hardware cost and achievable clock frequency can be evaluated by an FPGA implementation tool. A theoretical performance can be estimated by a logic simulation. In addition, to evaluate a practical performance including the actual effects such as the real memory latency and the on-chip bus conflict, we have developed a hardware platform on a real machine shown in Fig. 8. The Digilent ZYBO is an FPGA board that has ZYNQ FPGA (XC7Z010-1CLG400C) and 512MB DDR3 SDRAM. The ZYNQ has the Cortex A9 at 650MHz as the embedded processor (PS) and the reconfigurable fabric as Artix-7 FPGA (PL). The DDR3 SDRAM is mainly used for the frame buffers of the image. A CMOS camera (OmniVision OV9655) and a general purpose display supporting SXGA are attached to the ZYBO.

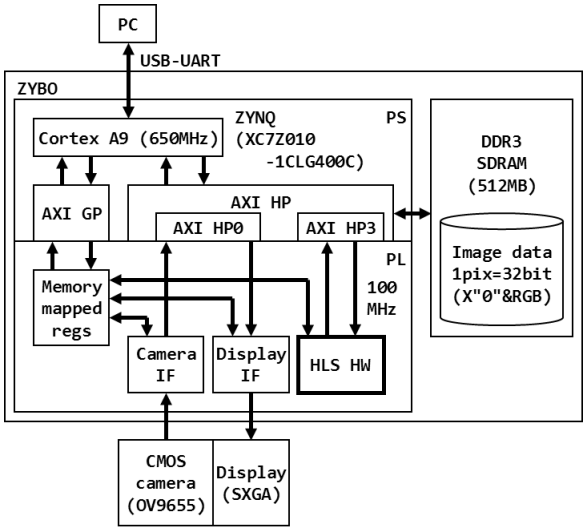


Fig. 8 Hardware platform.

On the hardware platform, the PL consists of some memory mapped registers via the AXI GP bus, the camera and display interfaces and HLS hardware connected to the AXI HP bus. The camera IF, the display IF and the HLS hardware can access the DDR3 SDRAM of the frame buffers as the AXI HP bus master through an implicit bus arbitration on the PS. The base addresses of the frame buffers on the DDR3 SDRAM can be specified by the memory mapped registers. To make the AXI HP bus masters easily share the same image, the pixel format of the image on the frame buffer is unified such that each pixel has 32bit width including 8bit R, G and B. All hardware on the PL run at 100MHz clock frequency.

For the HLS, FPGA implementation and logic simulation, we used Xilinx Vivado HLS 2016.4. This tool was launched on a personal computer with Intel Core i5-4570 at 3.20GHz and 16GB memory. The operating system was Microsoft 64bit Windows 10.

4.2 Case Study of Proposed C Description

To demonstrate and evaluate the characteristic features of our proposal, we apply our description method to a canny edge detection built on the chain of primitive image pro-

```
typedef struct {
  uint8_t mag; //Magnitude
  uint8_t dir; //Direction
} data;

 1: u32  ibuf[W*H],obuf[W*H];
 2: u8   gval[W*H],t1[W*H],t2[W*H];
 3: data t3  [W*H],t4[W*H];
 4: u8   t5  [W*H],t6[W*H],t7[W*H],t8[W*H],t9[W*H];
 5: u32  obuf[W*H];

 6: tiny_canny(u32 x[W*H], u32 y[W*H]){
 7: #pragma HLS INTERFACE m_axi port=x bundle=d
 8: #pragma HLS INTERFACE m_axi port=y bundle=d
 9: #pragma HLS DATAFLOW
10: #pragma HLS STREAM variable=ibuf dim=1
11: #pragma HLS STREAM variable=t1   dim=1
    ...
12: #pragma HLS STREAM variable=obuf dim=1
13:   memcpy  (ibuf,x   ,W*H*4);//for AXI burst
14:   gray    (ibuf,gval,W,H  );//Gray scaling(1D)
15:   smooth  (gval,t1  ,W,H  );//Gaussian filter(5x5)
16:   PF      (t1  ,t2  ,W,H,5,5);
17:   sobel   (t2  ,t3  ,W,H  );//Sobel filter(3x3)
18:   PF      (t3  ,t4  ,W,H,3,3);
19:   nmax_sup(t4  ,t5  ,W,H  );//Non-max. suppression(3x3)
20:   PF      (t5  ,t6  ,W,H,3,3);
21:   hyst_th (t6  ,t7  ,W,H  );//Hysteresis thresh.(1D)
22:   simp_th (t7  ,t8  ,W,H  );//Simple thinning(3x3)
23:   PF      (t8  ,t9  ,W,H,3,3);
24:   pix_gen (t9  ,obuf,W,H  );//Bin. to 32bit pixel(1D)
25:   memcpy  (y   ,obuf,W*H*4);//for AXI burst
26: }
```

Fig. 9 Simplified canny edge detection on vivado HLS tool.

cessing. Figure 9 depicts an overview of the C program list of the top function including sub-functions.

In Fig. 9, to specify an AXI bus master port supporting burst transfer, the pragma, `#pragma HLS INTERFACE...`, and `memcpy` are written to the arguments, `x` and `y`. They correspond to the pragma, `#pragma MEMBURST x, y`, shown in Fig. 1. To specify that each function runs in parallel synchronizing over a FIFO bridging the former sub-function and latter sub-function, the pragmas, `#pragma HLS DATAFLOW` and `#pragma HLS STREAM...`, are written. They correspond to the pragmas, `#pragma CONCURRENT` and `#pragma FIFO...`, shown in Fig. 1.

All sub-functions of the primitive image processing in the top function (`tiny_canny`) are made as fixed point version. The `gray` converts the input color pixel to the gray-scaled 8bit pixel in a 1D scanning fashion. The `smooth` smooths the gray-scaled image with a Gaussian filter using 5×5 window as shown in Fig. 4. The `sobel` performs the sobel filter using 3×3 window on the smoothed gray-scaled image as shown in Fig. 4. To simplify the hardware, we employed the absolute values to calculate the magnitude instead of the square and square root operations [8]. The edge direction was acquired by using constant value of arc tangent corresponding to the decline of the gradient calculated by the kernel of sobel filter. The directions were limited to 4. The `nmax_sup` performs the non-maxima suppression with 3×3 window [8] to extract only maxima points on the edge direction by eliminating the non-maxima points as shown in Fig. 4. Finally, the `hyst_th` of 1D scanning and `simple_th` of 2D scanning thin the edges constructed of the maxima

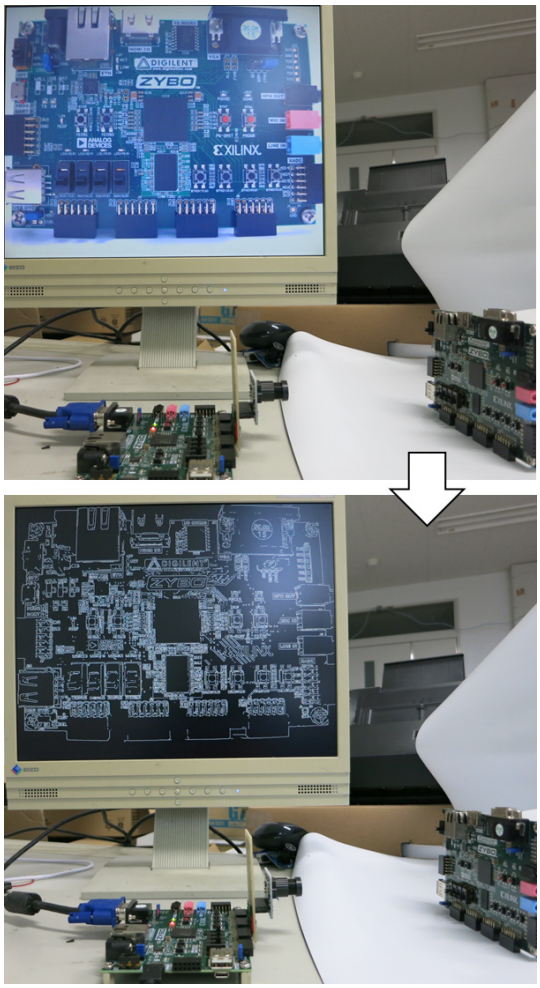


Fig. 10 Practical execution of canny edge detection (1280×1024).

points along the edge direction. The `hyst_th` performs hysteresis thresholding to mark the strong and weak edges using a low threshold and a high threshold [8]. The `simp_th` is a simplified thinning process using 3×3 window on the pixels marked by hysteresis thresholding. A full canny edge detection like OpenCV would perform a connectivity analysis to detect and link edges globally on the whole image using a global stack. The stacking access pattern of such algorithm is out of range for the assumption described in Sect. 2. Therefore, we approximated an edge thinning by eliminating the weak pixel without a strong pixel on 8 neighboring. The survive pixels are set to 1 as the true edges and eliminated other pixels are set to 0 as nothing. The binarized edged image is converted by `pix_gen` to the image data with 32bit pixel on the DDR3 SDRAM.

Note that the pixel feeders (`PF`) are inserted after the sub-functions performing 2D scanning.

Figure 10 is a picture showing a practical execution of the canny edge detection on ZYBO board. The upper side of Fig. 10 captures the board picture by the CMOS camera and passes though it to the display. The image size is 1280×1024. The lower side of Fig. 10 performs the devel-

Table 1 Hardware size and clock frequency with constraint of 9ns.

Proposed HLS HW (ZYNQ, xc7z010clg400-1)

Image [W*H]	320×240	640×480	1280×1024
LUT	2305	2341	2365
FF	2491	2541	2589
DSP	15	15	15
BRAM	18	18	18
CLK [MHz]	115	116	117

Conventional HLS HW (ZYNQ, xc7z010clg400-1)

Image [W*H]	320×240	640×480	1280×1024
LUT	1926	1942	1947
FF	2194	2224	2248
DSP	15	15	15
BRAM	13	13	13
CLK [MHz]	113	114	118

Straightforward HLS HW (Virtex7, xc7v2000tfhg1761-1)

Image [W*H]	320×240	640×480	1280×1024
LUT	2936	3905	
FF	2125	2314	
DSP	15	15	n/a
BRAM	574	1962	
CLK [MHz]	104	71	

oped canny edge detection to the captured image and outputs the edged image on the display.

4.3 Hardware Size and Clock Frequency

Table 1 shows the results of FPGA implementation after the place and route via Vivado HLS 2016.4. Before launching FPGA implementation, we set the constraint of the clock period to 9 ns. The top table shows the result of the canny edge detection reconstructed by our proposal shown in Fig. 9. The middle table means the result of the conventional reconstructed one removing PFs from Fig. 9. The bottom table indicates the result of the straightforward software without any software level reconstructions.

We implemented the proposed and conventional methods into the ZYNQ (xc7z010) considering the practical execution on the hardware platform shown in Fig. 8. While, we implemented the straightforward method into Virtex7 (xc7v2000t) due to the resource limitations of ZYNQ FPGA. For the pure software implementation to 1280 × 1024 image, we were not able to finish the implementation due to memory heap error reported by Vivado HLS 2016.4 on the used personal computer.

The proposed and conventional methods are converted to such hardware modules that show almost same amount of hardware regardless of the image size. In contrast, the straightforward method meaningfully increases the amount of hardware as the image size grows. Especially, the number of embedded memories (BRAMs) shows significant difference between the proposed method and the straightforward method. The proposed and conventional methods limit the parts dependent on the image size to the line buffers, connecting the sub-functions by the shallow FIFOs with 2 entries inferred as LUT-based shift registers[†]. In contrast, the straightforward method has inferred BRAMs instead of the FIFOs to communicate the processed image among the sub-functions. That is why the differences of the hardware size become bigger as the image size grows.

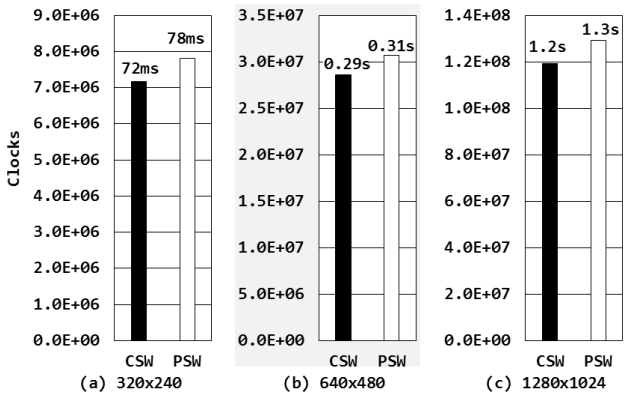


Fig. 11 Performance of software (CSW: conventional pure software, PSW: proposed reconstructed software).

Compared with the conventional one, our proposal increases the hardware size. This is because the proposal includes the PFs that the conventional method does not include. However, the PFs would be needed to prevent the deadlock across FIFO and perform pixel interpolation. Thus, this hardware investment is expected as valid. This aspect will be discussed in **4.5** in detail.

For the clock frequency, the proposed and conventional methods can achieve almost consistent clock frequency over the image sizes. The proposed and conventional methods construct the pipelined hardware on the whole hardware over several sub-functions. Thus, we think that the critical pass was cut well by the pipeline registers. In contrast, the straightforward method made a BRAM chain to construct the large embedded memories holding the processed image. Since this chain led a large critical pass, the straightforward one significantly degraded the clock frequency as the image size grows.

4.4 Performance Evaluation of Software

In HW/SW co-design, the decision of whether an image processing is implemented as software or hardware is the designer's responsibility through several trade-offs. That is, it is desired that the image processing software for the HLS can also achieve well performance as software processing on a used CPU compared with the pure software implementation so as not to intricately mix the HLS software and pure software.

Figure 11 shows the result of a comparative evaluation between the proposed software reconstructed for the HLS and the conventional pure software. We used Vivado SDK

[†]To make hardware size small as much as possible, we just set the depth of FIFO to 2. Effect of the FIFO depth to the performance is out of scope from this research.

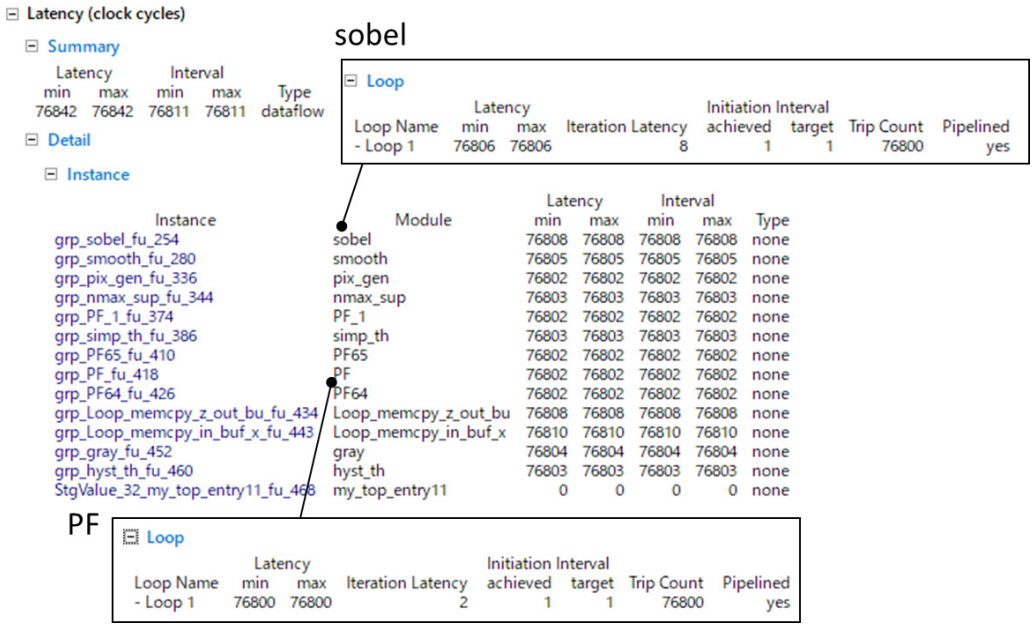


Fig. 12 Latency report of vivado HLS 2016.4.

2016.4 to compile the software by gcc with O2 option and executed them on the Cortex A9 processor on ZYNQ. The Cortex A9 used the instruction and data cache. The value of the bar graph is the averaged value of 10 times executions. The number of clocks was acquired from a 32bit performance counter which is one of the memory mapped registers we developed on the PL of ZYNQ. This counter runs at 100MHz clock frequency (clock period of 10 ns). Thus, the execution times shown on the top of the bar were calculated by multiplying the number of clocks with 10 ns.

The proposed software degraded the performance of about 8% to the conventional software. This is because some overheads such as shifting the buffers and registers and interpolating the number of pixels which do not exist in the conventional software have been introduced to the proposed software. However, the performance degradation is a relatively small such as 8%. This fact indicates that a unified image processing library using only HLS software executed on CPU or hardware module for HW/SW co-design can be established.

4.5 Performance Evaluation of Hardware

4.5.1 Latency Reported by High-Level Synthesis

The high-level synthesis tool, Vivado HLS 2016.4, attempts to generate the pipelined hardware where each sub-function has the pipelined structure with 1 of the initiation interval according to the pipeline pragma inserted. Table 2 shows the latency estimated by the high-level synthesis to the proposal, the conventional, and the straightforward methods.

The ideal throughput of the pipelined hardware is 1 pixel per clock. So, the number of pixels for each image size is shown as the ideal latency in the first row of the table. As the latency of each hardware is closer to the ideal latency, the HLS hardware has a well-pipelined structure.

Table 2 Latecny estimated by high-level synthesis [Clocks].

HW \ Image size	320× 240	640× 480	1280× 1024
Ideal	76800	307200	1310720
Proposed	76842	307242	1310762
Conventional	76838	307238	1310758
Straightforward	2111487	8523487	36539173

The latency of the proposed and conventional hardware is closer to the ideal latency. Thus, we think that they have well-pipelined structure. For example, sub-functions and PFs in the proposal are pipelined with 1 of the initiation interval as shown in Fig. 12. The reason why the proposed and conventional hardware show slightly longer latency than the ideal one is caused by the number of pipeline stages. Until the pipeline is filled by the pixels, the number of clocks same as the pipeline stages is added to the latency.

The latency of proposal is slightly larger than that of the conventional. This is because the PFs inserted into the proposal make the pipeline stage longer than that of the conventional. That is, compared with the conventional hardware, the PF dose not contribute the performance improvement although this difference of latency is very little as neglectable. Its contributions are to avoid the deadlock across FIFO and to perform the pixel interpolation implicitly.

As expected, the straightforward hardware shows the significant large latency compared with the proposed and conventional hardware. According to the report generated by Vivado HLS 2016.4, all attempts of pipelining were failed by resource conflict. Thus, the latency becomes significant larger than the proposed and conventional hardware with well-pipelined structure.

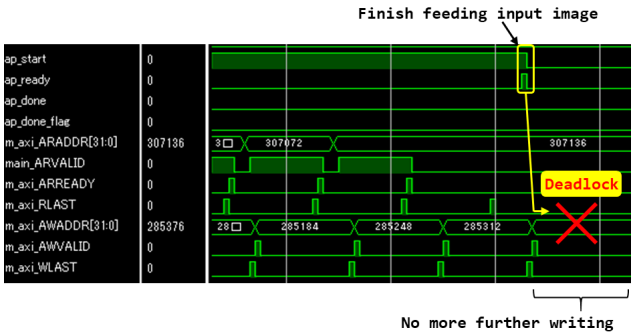


Fig. 13 Deadlock across FIFO on logic simulation.

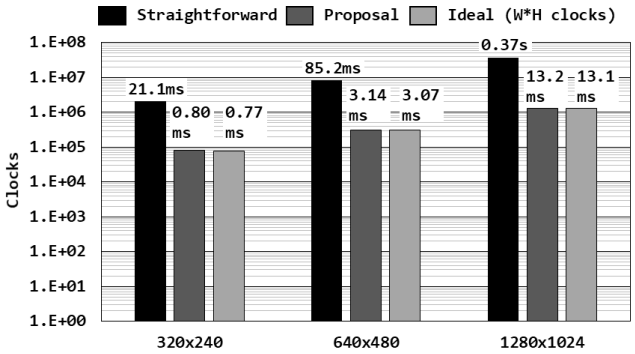


Fig. 14 Performance estimation of hardware via logic simulation.

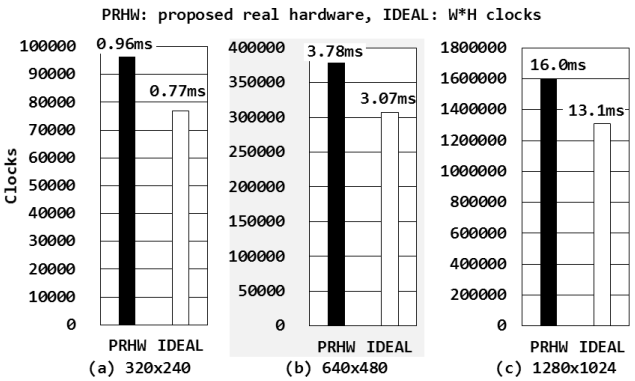


Fig. 15 Real performance of hardware on ZYBO.

4.5.2 Logic Simulation

To confirm whether our proposal can actually generate a well pipelined hardware of 1 pixel per clock although sub-functions are chained by the pixel feeders, we evaluated the performance of the HLS hardware on the logic simulation as a near ideal execution environment. The logic simulation was performed by the C/RTL co-simulation of Vivado HLS 2016.4.

In the logic simulation, the conventional hardware was not able to finish the execution correctly. Certainly the co-simulation on Vivado HLS returned the completion of the logic simulation of the conventional hardware. The co-simulation made the startup signal (ap_start) always valid through the whole simulation period. As a result, the hardware of the memory reading part was re-executed, the same image was fed again from first, and the finish signal (ap_done) was forced to '1'. That is, since the invalid pixels are pushed into the FIFO, the consumer that pops FIFO proceeded accidentally its own process using the pushed invalid pixels without deadlock. When using our test bench feeding the input image once, it has confirmed that the conventional hardware stalls at certain point as shown in Fig. 13. Also, we have confirmed that the same stall has occurred on the real FPGA board. This is because the deadlock across FIFO has occurred as expected. Thus, we would like to ignore the conventional hardware in the subsequent performance evaluations.

Figure 14 shows the result of the logic simulation for the proposed hardware, and the straightforward hardware as a reference. The straightforward hardware was generated from the pure software without any software level reconstruction while the proposed one was generated from the reconstructed software following our proposal. The number of clocks of the ideal hardware is equal to the number of all pixels of the image. The execution times shown on the top of the bar were calculated as well as those of Fig. 11.

Comparing the proposal and the ideal hardware, the proposed hardware can achieve almost ideal performance of 1 pixel per clock. This fact indicates that our proposal can be converted to a well pipelined hardware as expected. The little performance degradation to the ideal hardware about 1 to 4% is introduced by the Vivado HLS 2016.4 supporting the 16 burst transfer of the AXI bus. Every 16 burst transfer, the generated AXI interface outputs the valid addresses to the read and write channels. This overhead led to such little performance degradation.

To confirm the performance impact of our proposal on a real machine, we evaluated the performance on the hardware platform implemented into the ZYBO shown in Fig. 8. Figure 15 shows the result of the practical execution. Since the HLS hardware generated from the pure software without any software level reconstruction was not able to be implemented to the FPGA used, we evaluated only the HLS hardware generated from our software description on the real machine. The ideal number of clocks is same as that of Fig. 14. In addition, Table 3 indicates the speedup ratios of the hardware calculated by dividing the execution time of the software with that of the HLS hardware. In Table 3, the PRHW means the result of the real HLS hardware generated from our proposed software description. The SHW means the result of the HLS hardware generated from the pure software description without any software level reconstruction, which is the estimated value from the logic simulation as a reference.

The proposed real hardware shows the performance degradations of 22% to 25% than the ideal hardware. Comparing with the result of the logic simulation shown in Fig. 14, these degradations are larger. This is because the bus conflict between the read and write channels, the

Table 3 Speedup ratios calculated as SW/HW (CSW: conventional pure software, PSW: proposed software, PRHW: proposed real hardware, SHW: straightforward hardware estimated by logic simulation).

Image [W*H]	320×240		640×480		1280×1024	
SW / HW	CSW	PSW	CSW	PSW	CSW	PSW
PRHW	74.4	81.0	75.7	81.2	74.5	80.8
SHW (Est.)	3.39	3.69	3.36	3.60	3.26	3.53

resource conflict on the 16 bit data port of the DDR3 SDRAM and the characteristic of DDR3 SDRAM such as bank switching and refreshing might badly affect the performance. These practical influences are not considered by the C/RTL co-simulation on the Vivado HLS 2016.4. However, as shown in Table 3, the proposed real hardware was able to achieve a good performance of more than 70 times to the software execution although there are many influences in the real environment degrading the performance.

The straightforward hardware can also improve the performance of image processing by converting to the HLS hardware as shown in Table 3. However, the results indicate that our proposed describing method can extract the high performance nature of the hardware significantly.

5. Conclusion

This paper has shown a describing method of an image processing software in C for a high-level synthesis (HLS) technology to realize an efficient hardware, considering function chaining. According to the proposed method, any number of sub-functions can be chained, maintaining the pipeline structure over the HLS hardware. In addition, the deadlock due to the mismatch of the number of pushes and pops on the FIFO connecting the functions is eliminated and the interpolation of the border pixels is done implicitly. Thus, it is expected that an image processing hardware to which the HLS technology converts our proposed description can achieve a near ideal performance of 1 pixel per clock although the chain of sub-functions is even long.

The case study of a canny edge detection on ZYNQ FPGA demonstrates that our proposal can easily describe a top function to be converted to the expected hardware mentioned above. The experimental results show that our proposal can be converted to a well-pipelined hardware with a moderate size then achieve a performance gain of more than 70 times compared with a software execution.

As future work, we will apply our proposed describing method to more applications and evaluate them. In addition, we will develop a general purpose image processing library including the HLS software following our proposal.

References

[1] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y.T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of fpga high-level synthesis tools," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.35, no.10, pp.1591–1604, 2016.

[2] D.G. Bailey, "The advantages and limitations of high level synthesis for fpga based image processing," Proceedings of the 9th International Conference on Distributed Smart Cameras, pp.134–139, 2015.

[3] M. Schmid, N. Apelt, F. Hannig, and J. Teich, "An image processing library for c-based high-level synthesis," Proceedings of 24th International Conference on Field Programmable Logic and Applications, pp.1–4, 2014.

[4] D. Bagni, "Median filter and sorting network for video processing with vivado hls," Xcell Journal, vol.86, pp.20–28, 2014.

[5] J. Monson, M. Wirthlin, and B.L. Hutchings, "Optimization techniques for a high level synthesis implementation of the sobel filter," Proceedings of International Conference on Reconfigurable Computing and FPGAs, pp.1–6, 2013.

[6] E. Kalali and I. Hamzaoglu, "Fpga implementations of hevc inverse dct using high-level synthesis," Proceedings of Conference on Design and Architectures for Signal and Image Processing, pp.1–6, 2015.

[7] A. Yamawaki and M. Iwane, "High-level synthesis method using semi-programmable hardware for c program with memory access," Engineering Letters, vol.19, no.1, pp.50–56, 2011.

[8] B.R. Masters, R.C. Gonzalez, and R.E. Woods, "Digital Image Processing Third Edition," J. Biomed. Opt., vol.14, no.2, 2009.

[9] J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelley, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, and P. Hanrahan, "Darkroom: Compiling high-level image processing code into hardware pipelines," ACM Trans. Graph., vol.33, no.4, pp.144:1–144:11, 2014.

[10] N. Chugh, V. Vasista, S. Purini, and U. Bondhugula, "A dsl compiler for accelerating image processing pipelines on fpgas," Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, pp.327–338, 2016.

[11] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," Proceedings of the IEEE, vol.100, pp.1411–1430, 2012.

[12] Xilinx, "Vivado design suite user guide high-level synthesis." UG902 (v2016.4) Nov. 30, 2016.

[13] S. Cass, "The 2016 top programming languages - c is no.1, but big data is still the big winner." http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016, July 2016.

[14] M. Ekstrom, Digital Image Processing Techniques 1st Edition, Elsevier, 1984.

[15] S. Kyo, S. Okazaki, and T. Arai, "An integrated memory array processor architecture for embedded image recognition systems," 32nd International Symposium on Computer Architecture (ISCA'05), pp.134–145, 2005.

Akira Yamawaki received the B.S. and M.S. degrees in Electrical Engineering from Kyushu Institute of Technology in 1997 and 1999, respectively. During 1999–2000, he stayed in Mitsubishi Electric Company. From 2000–2015, he was an Assistant Professor at the Kyushu Institute of Technology. He received the Ph.D. degree in Electronic Engineering from Kyushu Institute of Technology in 2006. Since 2015, he has been an Associate Professor at the Kyushu Institute of Technology. His current research interests include digital hardware system, smart sensor system, sensor network and reconfigurable hardware system. He is a member of IEEE, IEICE and IIAE.

Seiichi Serikawa received the B.S. and M.S. degrees in Electronic Engineering from Kumamoto University in 1984 and 1986. During 1986-1990, he stayed in Tokyo Electron Company. He received the Ph.D. degree in Electronic Engineering from Kyushu Institute of Technology in 1994. From 1994 to 2000, he was an Assistant Professor at the Kyushu Institute of Technology. From 2000 to 2004, he was an Associate Professor at the Kyushu Institute of Technology. Science 2004, he has been a Professor at the Kyushu Institute of Technology. Recently, he is a Vice President of Kyushu Institute of Technology. His current research interests include computer vision, sensors, and robotics. He is a member of IEEJ, IEICE, and the president of IIAE.