Performance Improvement of Hardware by Series Duplicating Data Buffer for High-level Synthesis

Moena Yamasaki^a and Akira Yamawaki^{a, *}

^aKyushu Institute of Technology, 1-1 Sensui, Tobata, Kitakyushu 804-8550, Japan

*Corresponding Author: yama@ecs.kyutech.ac.jp

Abstract

To occupy appropriate the expanding market share of embedded image processing systems, it is important to quickly develop and launch a high-performance and low-power product onto the market tracking the short life cycle of recent products. To achieve high performance and low-power produce quickly, it is effective to develop the hardware module of high computational software processing using high-level synthesis technology automatically converting software to hardware. However, high-level synthesis cannot convert software not taking into hardware organization to the efficient hardware with high-performance and low-power. This paper proposes a software description method for high-level synthesis that replicates histograms in series and pipelines the pre and post processing across the histogram. The experimental result on the real machine demonstrates the effects of the proposed method.

Keywords: high-level synthesis, HLS, buffer, overlap

1. Introduction

For rapidly realizing high-performance and low-power embedded image processing products, it is effective to implement image processing hardware using high-level synthesis technology. High-level synthesis (HLS) is a technology that automatically converts software to hardware, and can reduce the burden of hardware development. However, high-level synthesis of software programs that do not consider the hardware configuration cannot be converted to high-performance hardware [1-2].

For example, there are image processing consisting of the pre-processing and post processing across the buffer. The pre-process produces data to a buffer. The post-process

consumes data pre-process has generated in the buffer. High-level synthesis inserts a synchronization mechanism with a waiting time between the pre-process and post-process to resolve data dependencies on the buffer. This synchronization generates the pipeline stalls into the pipelined pre-process and post-process. Thus, the hardware cannot show the inherent high-performance nature of the hardware.

In this paper, we propose a software program description method for high-level synthesis in order to solve the problem of inserting such waiting time. Then, the effect of the proposed description method is shown through experiments.

The rest of the paper is organized as follows, Section 2 clarifies the problems of data-dependent buffers. Then, in Section 3, we describe the description method we propose, and in Section 4, we show applications that apply the description method. Section 5 examines the experimental results, and finally, Section 6 concludes this paper.

2. Problem definition of high-level synthesis

2.1 Processing flow including data dependent buffer

One of the simplest organizations of image processing

Fig.1. Processing flow for high-level synthesis

Fig.2. Unsatisfactory processing flow of high-level synthesis

Fig.3. Duplicate execution of pre-processing

consists of the pre-process and the post-process across the buffer. The pre-process produces data into the buffer. The post-process consumes data in the buffer produced by the pre-process.

Fig. 1 shows an execution snapshot of such image processing consisting of the pre-process, buffer, and post-process. High-level synthesis is good at handling the processing flow as shown in Fig. 1. That is, the post-process continuously uses the data every when the pre-process produces data. In this case, high-level synthesis can pipeline the entire process over the pre-process and post-process. As a result, the automatically generated hardware has an ideal pipeline configuration can provide the output data per one clock. Examples of such processing flows include image binarization with fixed thresholds, image compression with run length, and color space conversion of images. The buffer is instantiated as a register or a FIFO whose role is simply flowing data continuously.

However, as shown in Fig. 2, there are some image processing flows which the high-level synthesis cannot handle well. In such flow, the post-process cannot start operation unless the pre-process has generated all the data. For example, there is automatic binarization by discriminant analysis and contrast improvement by flattening the luminance histogram. Buffers are usually

realized by using embedded memory.

The pre and post processes must use the buffer exclusively each other to avoid the data corruption caused by the data dependency. The post process cannot consume the buffer before the pre-process finishes producing the data. The pre-process cannot update the buffer before the post process finishes consuming the buffer. Especially, this problem becomes significant to resolve for the continuous image processing like movie.

In this case, high-level synthesis cannot be pipelined for the entire process. Therefore, the pre-process and the post-process wait for each other to complete the use of the buffer. Therefore, ideal hardware providing the output data per one clock cannot be generated.

2.2 Parallel execution of pre and post-process

High-level synthesis tools generally have a pragma specifying to make inner processes executed independently in a parallel fashion. Therefore, if a pragma of parallel execution is added to the pre-process and post-process shown in Fig. 2, each process may independently run on the hardware. As a result, high performance is expected due to the overlapped execution between per and post processes across the buffer boundary as shown in Fig. 3.

The pre-process outputs all data into the buffer. Then,

Fig.4. Hardware logic simulation using pragma directives

the post-process uses the buffer and outputs the final data. Note that the pre-process can be generating the next data to the buffer while the post process is performing its job using the previous data on the buffer. That is, the pre-process and post-process can be overlapped to improve the performance of continuous image processing.

However, existing high-level synthesis tools cannot automatically generate hardware that achieves such mechanism overlapping the pre and post-processes as shown in Fig.3 by only using pragmas.

For example, a high-level synthesis tool of Xilinx Vivado HLS 2017.4 has the pragma, DATAFLOW, that can specify the pre and post processes to operate independently. Fig.4 shows the operation waveforms of the hardware logic simulation. In order to operate the former stage and the latter stage independently, a flag indicating the completion of each process is needed. However, only the completion flag, ap_done, of the entire process including pre and post processes is only present in this waveform. This fact indicates that high-level synthesis tool using only pragmas cannot generate the expected hardware whose pre and post processes are completely separated.

3. Proposed method

3.1 Proposed hardware configuration

To make high-level synthesis generate well organized high-performance hardware, it is necessary to input not the pure software just coded from the algorithm but the software well reconstructed with consideration of the hardware organization.

In the case of the processing flow shown in Fig. 2, high-level synthesis cannot realize an ideal pipelined hardware for the entire process due to the data dependency on the buffer at all. Fig. 3 shows another method for improving performance instead of using a pipeline. In this

Fig.5. Proposed hardware configuration

method, the pre-process and post-process are separated and overlapped. Pure software needs to be rebuilt so that such hardware can be generated by high-level synthesis.

In reconstructing the software, consider the hardware configuration shown in Fig. 5. In this configuration, buffers that had a data-dependent relationship between the pre-processing and post-processing are duplicated in series, and each is locally stored in the pre-processing and post-processing. The pre-processing reads data from the input port (**src1**) and writes the processed data to the local buffer (**lb**) according to its own processing. After that, the contents of the local buffer are continuously copied to the local buffer in the post-processing. When the duplication of the local buffer is completed, the post-processing reads the data from the input port (**src2**), proceeds with its own processing while continuing to use the contents of the duplicated local buffer, and sends the processed data to the output port (**dst**).

3.2 Program description method

Fig.6 shows an overview of the software description that maps the hardware configuration shown in Fig. 5. The first to fifth lines in Fig. 6 are the top functions including the pre-stage processing (**pre**) and the post-stage processing (**post**). The pre-processing corresponds to lines 6 to 12, and the post-processing corresponds to lines 13 to 20. The pragmas in the figure are described according to

```

1:top(*src1,*src2,*dst){
2:  BUFF b[BN];
3:  pre (src1, b    );
4:  post(src2, b, dst);
5:}
6:pre(*src, *b){
7:#pragma HLS interface=m_axi port=src1 bundle=d0
8:#pragma HLS interface=axis port=b    bundle=buf
9:  BUFF lb[BN];
10: pre_core(src, lb);
11: for(i=0;i<BN;i++) b[i] = lb[i];
12:}
13:post(*src, *b, *dst){
14:#pragma HLS interface=m_axi port=src2 bundle=d1
15:#pragma HLS interface=m_axi port=dst  bundle=d0
16:#pragma HLS interface=axis port=b    bundle=buf
17:  BUFF lb[BN];
18:  for(i=0;i<BN;i++) lb[i] = b[i];
19:  post_core(src, b, dst);
20:}

```

Fig.6. Proposed software configuration

Xilinx's Vivado HLS. Considering large-scale input / output images, the image data is assumed to exist in a large-capacity external memory such as DDR SDRAM.

The first pragma in the first stage of processing assigns the input data arguments to physical ports. At that time, the interface (**m_axi**) to the external memory is specified. The following pragma specifies the local buffer write port as the continuous (stream) data interface. The internal processing performs processing (**pre_core**) to create a local buffer from input data and continuously writes it to the outside.

The first and second pragmas in the post-processing assign input and output data arguments to physical ports. At that time, the interface of the external memory is specified. The third pragma specifies a stream interface to continuously receive local buffer writes from the previous stage. The internal processing copies the contents of the local buffer received from the previous processing to its own local buffer. Subsequent processing (**post_core**) creates output data using the local buffer and input data.

3.3 Expected behavior

Using the proposed method described in the previous section, the high-level synthesis tool can generate hardware in which the pre-processing and post-processing operate independently. Fig.7 shows an overview of the expected overlapping behavior when the hardware generated by performing high-level synthesis separately for the pre-processing and post-processing is integrated.

In T1, the pre-processing reads the input data and generates data in its own local buffer.

At T2, the preceding local buffer was copied to the

Fig.7. Duplicate execution

succeeding local buffer.

At T3, the post-processing generates output data using its own local buffer. At the same time, the pre-processing reads the next input data and updates its own local buffer. At this time, the pre-processing and post-processing are operating simultaneously and overlapping. After that, buffer duplication at T4 and parallel execution of the pre-stage and post-stage of T5 are performed.

In the proposed method, a waiting time occurs at the time of buffer duplication like T3 and T4. However, when the replication time is shorter than the time of the pre-processing and the post-processing, data input / output is performed almost without interruption, and good performance improvement can be achieved by the parallel operation of the pre- and post-processing.

4. Application to image processing

Automatic binarization using the discriminant analysis method (Otsu method) [3] is an example of image processing to which the proposed description method is applied.

Automatic binarization includes a process of creating a histogram of luminance values and the number of pixels from an input image, and a process of performing binarization by performing discriminant analysis using the histogram. The former is defined as pre-processing, and the latter is defined as post-processing. The value of the histogram depends on the data, and a waiting time occurs between the previous and next stages.

Fig.8. Process flow of automatic binarization

Fig.8 shows the processing flow that applies the proposed description method. We also incorporate the method we have developed up to now [4-5]. A program was written to have a local histogram by duplicating the histogram in series so that when it was put into high-level synthesis, the hardware would be independent at the front and rear stages.

5. Experiment and Discussion

5.1 Experimental Setup

To perform the experiment on the real machine, we have developed the hardware platform. The used FPGA is Xilinx ZYNQ FPGA. The used FPGA board is Digilent ZYBO. The embedded processor of ARM Cortex-A9 runs at 650MHz. The hardware modules run at 100MHz.

The software program of the proposed description method shown in Chapter 4 was converted to hardware (HDL program) by Vivado HLS 2017.4 of Xilinx HLS tool. Next, the generated HDL program was implemented on the FPGA board Digilent ZYBO (Zynq-7000 development

Fig.9. Hardware amount

board), and the hardware was evaluated. The size of the input image for automatic binarization is 256×128 pixels. The same image repeatedly inputted to imitate the video environment.

5.2 Hardware Amount

Fig.9 shows a graph of the number of FFs and LUTs, which is the amount of hardware. In the proposed method, since the histogram was duplicated, the amount of hardware increased, but it increased slightly by 0.02%.

5.3 Execution time

Fig.10 shows the execution times of the conventional method and the proposed method measured by FPGA. The result measured by the FPGA is denoted by HW, and the result measured by the embedded processor in the FPGA board is denoted by SW. The operating frequency of the FPGA is 100MHz, and the embedded processor is 650MHz. Since the difference between the conventional method and the proposed method of SW was about 0.001 ms, only the proposed method was shown in the graph.

Focusing on the HW graph, with 16 images, the proposed method reduced 20.6% compared to the conventional method. This indicates that the speed was increased by parallelizing the former stage and the latter stage.

5.4 Power improvement ratio

Since the operating frequency of the embedded processor (SW) and that of the FPGA (HW) are different, the power improvement ratio considering the operating frequency is calculated and shown in Fig. 11. This graph shows that the larger the value, the more power-saving HW is compared to SW. With 16 images, the proposed method improved 1.23 times compared to the conventional method.

From the above, it can be said that the proposed method is an effective description method because the

Fig.10. Execution time

Fig.11. Power improvement ratio

power consumption is increased by 1.23 times with the hardware amount increased by 0.02% compared with the conventional method.

6. Conclusions

In this paper, we have described a software program description method for high-level synthesis, in which the pre-stage and post-stage processes are parallelized and the performance is improved by duplication of processes with buffer boundaries.

As a result of adopting the proposed method to the automatic binarization processing of the discriminant analysis method, it was possible to improve power by 1.23 times by increasing the hardware amount by 0.02%. From this, it can be said that the pre-processing that generates data for the buffer and the post-processing that consumes data in the buffer are effective description methods for image processing.

We will continue to develop various software description methods that enable high-level synthesis to generate high-performance hardware.

References

- (1) R. Nane, V.-M. Sima, B. Olivier, R. Meeuws, Y. Yankova, and K. Bertels, "DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler", In FPL, pp. 619-622. 2012.
- (2) C. Pilato and F. Ferrandi, "Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications", In FPL, pp. 1-4, 2013.

- (3) Rafael C. Gonzalez and Richard E. Woods, "Digital Image Processing Third Edition," Pearson Prentice Hall, pp. 743-747, 2008.
- (4) Moena Yamasaki, Satoshi Yonemitsu and Akira Yamawaki, "Effect of Redundant Function Execution to Reduce Memory Access on High-level Synthesis," Proceedings of the 6th IIAE International Conference on Industrial Application Engineering 2018, pp.206-209, 2018.
- (5) Moena Yamasaki and Akira Yamawaki, "Description Method for High-level Synthesis of Histogram Generation and Their Evaluation," IEIE Transactions on Smart Processing and Computing, vol. 8, no. 3, pp.178-185, June 2019.