Introducing Semi-programmable Hardware to a Real High-Level Synthesis Tool

Akira Yamawaki * Seiichi Serikawa † Masahiko Iwane ‡

Abstract– **The semi-programmable hardware is a design-level hardware architecture residing on the pass where C program with memory access is converted to hardware. The SPHW realizes the memory access controller and buffer by writing the software program and parameters respectively. Compared with the cases that use only HDL, the SPHW which can design the efficient memory controllers at C-level abstraction reduces the development time significantly. In addition, the SPHW shows the comparable performance compared to the HDL hardware containing the custom memory controller even if it is written at high-abstraction in the SPHW. In general, the high-level synthesis tool whose design entry is C program is often used to reduce the burden designing the data processing hardware. However, the SPHW has not been introduced into any HLS technology yet. This paper develops the true C level-design environment including the SPHW as the data processing hardware on a real commercial HLS tool, Handel-C. By using the SPHW providing the register-based data interface to the data processing hardware, we demonstrate that the HLS tool can easily write the hardware accessing to the memory in C language. This is because this interface hides the detail of the memory devices and the memory access patterns, by providing the data processing hardware with the simple stream data. For hiding memory access latency, the simple software-pipelining can be applyed to the memory access program and parameters of the buffer. Consequently, the designer can realize the data processing hardware with data-prefetching mechanism at the complete C-level design entry.**

Keywords: high-level synthesis, hardware design, memory latency hiding, system-on-chip, hardware architecture

1 Introduction

For the design of the system-on-chips, the high-level synthesis (HLS) technologies generating the hardware from C program have been researched and developed [1–7]. The HLS tool can reduce the design burden significantly due to the high design abstraction. Generally, the HLS technologies are good at generating an efficient data processing hardware assuming simple and typical data access patterns like the stream data. For example, some compilers of the HLS support only the dedicated memory access pattern [4, 6, 7]. Across the users, the application programs and the buffering methods, the memory access patterns are different. Thus, the memory accesses are hard to be treated systematically by an algorithmic way. In addition, the memory access latency cannot be hidden by the data prefetching [8] implicitly [1–7]. To hide the memory latency, the hardware has to be written skillfully in the C description with the deep knowledge of the used HLS tool and target device. As a result, the HLS tool might generate the hardware including an efficient memory access controller. Even if the HLS tool is used, such burden may be comparable to designing a custom memory access controller from scratch in a hardware description language (HDL).

To tackle the problems mentioned above, we have proposed a design-level hardware architecture (semi-programmable hardware: SPHW) which is inserted onto the pass converting the C program to the hardware [9]. The SPHW realizes the memory access controller and the buffer by writing the software program and parameters respectively. Compared with the design cases that use the HDL, the SPHW which can design the efficient memory controllers at C-level abstraction reduces the development time significantly [9]. In addition, the SPHW shows the comparable performance compared to the HDL hardware containing the custom memory controller even if it is written at high-level abstraction [9].

However, the SPHW has not been introduced into any HLS technology yet. This paper attempts to realize the complete C level-design environment including the SPHW on a real commercial HLS tool, Handel-C. By using the SPHW providing the register-based data accessing interface, we demonstrate that the HLS tool can easily write the hardware accessing to the memory in C. This is because this interface hides the detail of the memory devices and the memory access patterns, by providing the data processing hardware with the simple stream data. By introducing the SPHW into the HLS tool, the simple

*This research was partially supported by the Kayamori Foundation of Informational Science Advancement and the Grant-in-Aid for Young Scientists (B) (22700055). Faculty of Engineering, Kyushu Institute of Technology, 1-1, Sensui, Tobata, Kitakyushu, Fukuoka 804-8550 Japan. Email: yama@ecs.kyutech.ac.jp

†Faculty of Engineering, Kyushu Institute of Technology, 1-1, Sensui, Tobata, Kitakyushu, Fukuoka 804-8550 Japan.

‡Adviser of Yuundo Co.,Ltd. Japan.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

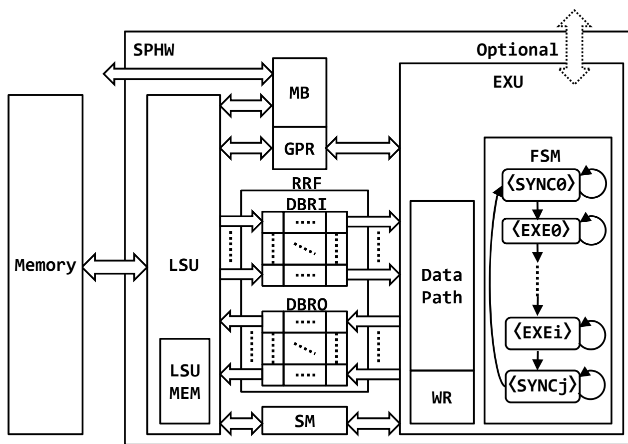

Figure 1: Block Diagram of SPHW

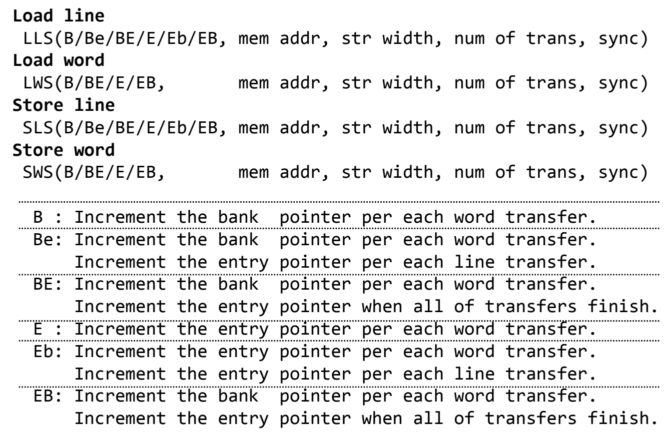

Figure 2: Load/Store instructions of LSU

software-pipelining with double buffering to hide memory access latency is easily implemented into the data processing hardware at high-level abstraction.

The rest of the paper is organized as follows. Section 2 describes the overview of the SPHW. Section 3 shows the design flow using the SPHW. Section 4 demonstrates the SPHW mapping by using an example of the color conversion from RGB to YCrCb. Section 5 performs some preliminary experiments and shows the experimental results. Finally, Section 6 concludes the paper.

2 SPHW Architecture

2.1 Organization

Fig. 1 shows the organization of the SPHW. The load/store unit (LSU) transfers the data between memory and reconfigurable register file (RRF). The programs to be executed by the LSU are stored in the LSU memory (LSUMEM). The execution unit (EXU) is data processing hardware. The synchronization mechanism (SM) performs the producer-consumer synchronization between the LSU and the EXU. The producer performs the release synchronization to invoke the consumer waiting for the data on the RRF by the wait synchronization.

The reconfigurable register file (RRF) consists of the input/output data buffer registers (DBRI and DBRO). They have one or more banks which contain one or more entries. The number of the banks and entries are configurable by the parameters. Thus, the suitable buffer for the data processing hardware on the EXU can be implemented by parameters. The mailbox (MB) is control/status registers for the SPHW. The external modules can check the statuses of the SPHW via mailboxes. The parameters required for the SPHW execution can be set via the mailboxes. The general purpose register (GPR) is used by the LSU and the EXU.

The EXU has the finite state machine (FSM), the working registers (WR) and the data path. The FSM has the states ($\langle \mathrm{EXE_i} \rangle$) to control the data path. In addition, the states ($\langle \mathrm{SYNC_i} \rangle$) to synchronize the LSU are inserted.

2.2 Memory Access

The LSU has the load/store instructions per the word and the line containing continuous words as shown in Fig. 2. Each of instructions can specify the number of transfers and the stride width per each word/line transfer. That is, the LSU can perform the gather/scatter operations by one instruction. Since the load/store instructions have the synchronization field, the synchronization can be also performed simultaneously with the memory access. The pointers to the bank and entry can be incremented automatically according to the notation in the instruction. The LSU converts the distributed data in the memory to the streamed data in the RRF, executing the sophisticated load/store instructions.

Fig. 3 (a) shows the examples implementing a double buffer for the streaming data. We assume that the line contains 4 words, the number of banks of the DBRI is 8 and each bank contains 2 entries. Fig. 3 (b) shows an example loading the 4×4 window. In this example, the DBRI has 4 banks containing 4 entries. As shown in these examples, the LSU can easily realize the sophisticated memory accesses.

3 SPHW Design Flow

Fig. 4 shows the framework of the design flow which employs the SPHW. In this case, we employ the Handel-C [1] as the HLS tool for the EXU.

For the SPHW, the memory accesses are implemented by the software programming to the load/store unit (LSU). The reconfigurable register file (RRF) is configured by the parameters to implement the optimum buffer. The data processing unit (EXU) streamly processes the sequential data on the RRF. The memory data which shows the

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

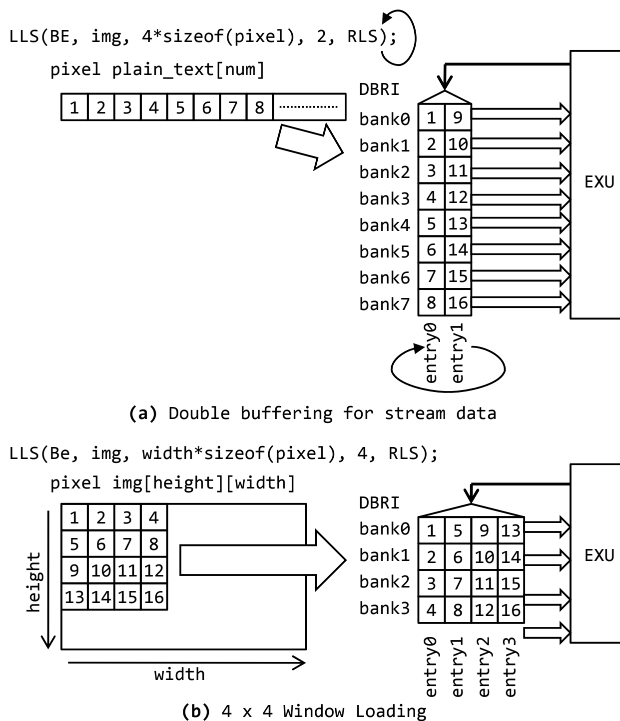

Figure 3: Example of Load Instruction

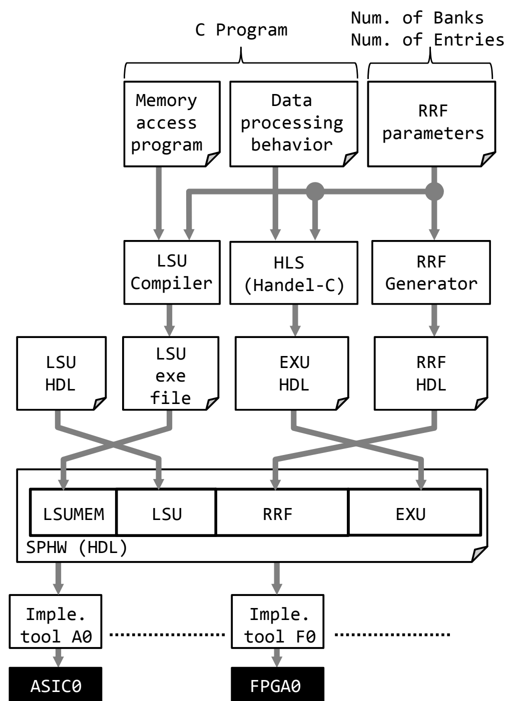

Figure 4: SPHW Design Flow When Using Handel-C

sophisticated access patterns are put into the RRF as the stream data by the LSU.

The Handel-C is based on the concept of the communicating sequential processes (CSP) model [10] whose input/output are the streaming interface. Thus, the Handel-C hardware is a good candidate as the EXU to be connected to the RRF [1].

Since the LSU and the EXU are executing individually across the RRF, the memory access by the LSU can overlap onto the data process by the EXU. By using the SPHW, the designer can design the hardware with the data prefetching mechanism [8] easily in the high-level description using the program and parameters.

4 Mapping Example

Fig. 5 and Fig. 6 show an example of mapping the color conversion from RGB to YCrCb into the SPHW. The former writes the hardware behavior in Handel-C. The latter writes the LSU program in C-like language. We have developed the tool converting the LSU program to the machine code by perl.

Now, we assume that the pixel of the image data is 32bit containing each of 8bit-R, G and B data. In this version, the SPHW supports the following features.

[1]Most HLS tool is good at converting the stream processing to the hardware [1–7].

(1) The number of words in the line is 4. The word width is 32bit.

(2) The width of each bank of the DBRI/O is 32bit.

(3) The LSU supports the burst transfer containing 4 words.

(4) The LSU is the pipelined scalar processor with 3 stages.

Fig. 5 shows an overview of mapping to the SPHW. The RGB data in the memory is loaded by the LSU into the DBRI. The EXU waits until the RGB data needed is stored into the DBRI. When the LSU loads the RGB data and releases the EXU, the EXU starts to process the RGB data in the DBRI and stores YCrCb data into the DBRO. Then the EXU releases the LSU waiting the YCrCb data. After this, the LSU stores the YCrCb data into the memory.

Fig. 6 shows the LSU programming. In this case, the mailbox #0 (MB0) is used as the start flag invoking the SPHW. The read address of the RGB data and write address of the YCrCb data are set to the MB1 and MB2. The stride width per line transfer that is 16byte is set to the MB3. The number of the transferred lines to fill all banks is set to MB4, which divides the number of banks by 4. The number of total lines over the image data is set to the MB5, which divides the number of total pixels by the number of banks. The MB6 is the flag indicating that the SPHW finishes.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

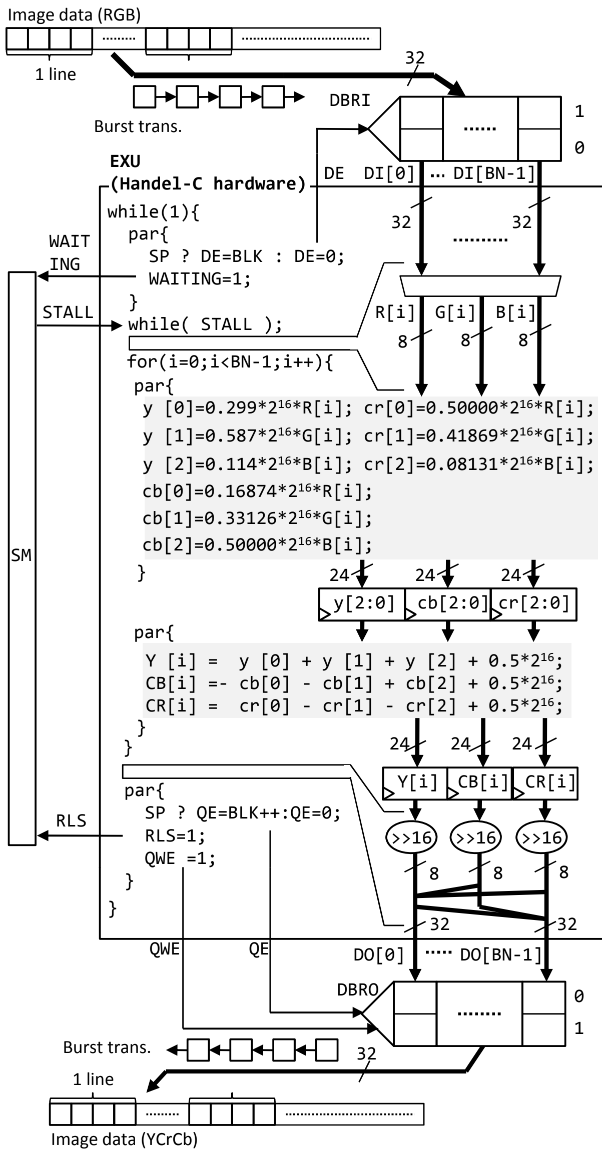

Figure 5: Mapping Image from RGB2YCrCb to EXU.

Fig. 6 (a) is the straight-forward programming. The LSU waits by spin-lock on the MB0 until it is set to 1. Then, the LSU resets the MB0 as the start flag and resets the MB1 as the end flag. The LSU loads the lines into all banks and performs the release synchronization (RLS). Then, the LSU performs the wait synchronization (WAIT) and stores the processed lines in the DBRO into the memory. This program is very simple and intuitive but suffers from the memory access latency.

Fig. 6 (b) is the LSU program of which the software pipelining [8] is applied to hide the memory access latency. In the software pipelining, the load instruction (LLS) and the store instruction (SLS) in the main loop are copied to the front of the main loop and the back of it respectively. In the main loop, the data used at the next iteration is loaded at the current iteration. Thus, the memory accesses of the LSU are overlapped with the

```
-- MB0 : Start flag
-- MB1 : Read  address
-- MB2 : Write address
-- MB3 : Stride width (16)
-- MB4 : Number of burst trans.
--       = (num. of banks) / 4
-- MB5 : Number of total trans.
--       = (num. of pixels) / banks
-- MB6 : End flag
do{
  while( MB0 == R0 );
  MB0=0; MB6=0;
  do{
    LLS(B, MB1, MB3, MB4, RLS );
    SLS(B, MB2, MB3, MB4, WAIT);
    R2 = R2 + R1;
  }while( MB5 > R2 )
  MB6 = 1;
}while(1);
```

(a) Straight-forward LSU program

```
do{
  while(MB0 == R0);
  MB0=0; MB6=0;
  LLS(BE, MB1, MB3, MB4,RLS);
  R2++;                  ↑ copy
  do{
    LLS(BE, MB1, MB3, MB4, RLS );
    SLS(BE, MB2, MB3, MB4, WAIT);
    R2++;                ↓ copy
  }while( MB5 > R2 );
  SLS(BE, MB2, MB3, MB4, WAIT);
  MB6 = 1;
}while(1);
```

(b) Software-pipelined LSU program

Figure 6: LSU Programming of RGB2YCrCb.

data processing of the EXU. For the EXU, as shown in Fig. 6 (a), the number of entries of the DBRI/O becomes two by denoting the SP as 1. In each iteration in the main loop, the entry pointer (BLK) is toggled. As mentioned above, the double buffering can be implemented easily.

5 Experiment and Discussion

Varying the number of banks, we implement the SPHW shown in Fig. 5 into the Virtex5 FPGA of which the speed grade is 10. We used the ISE12.1 in implementation.

Fig. 7 shows the number of LUTs and FFs. The NoSP means the straight-forward mapping and the SP means the software-pipelined mapping to overlap the memory access with the data processing. The result shows that the SP does not affect the circuit size despite fact the number of the bank entries is doubled compared to the NoSP. This is because each LUT used to the bank of the DBRI and DBRO can contain 32 entries. Thus, until the number of entries exceeds 32, the circuit sizes of the SP and NoSP are same. In contrast, as the number of the banks (LUTs) increases, the circuit size also increases. For the EXU, the increase of the banks increases the temporal registers (FFs) used as shown in Fig. 5. Since the

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

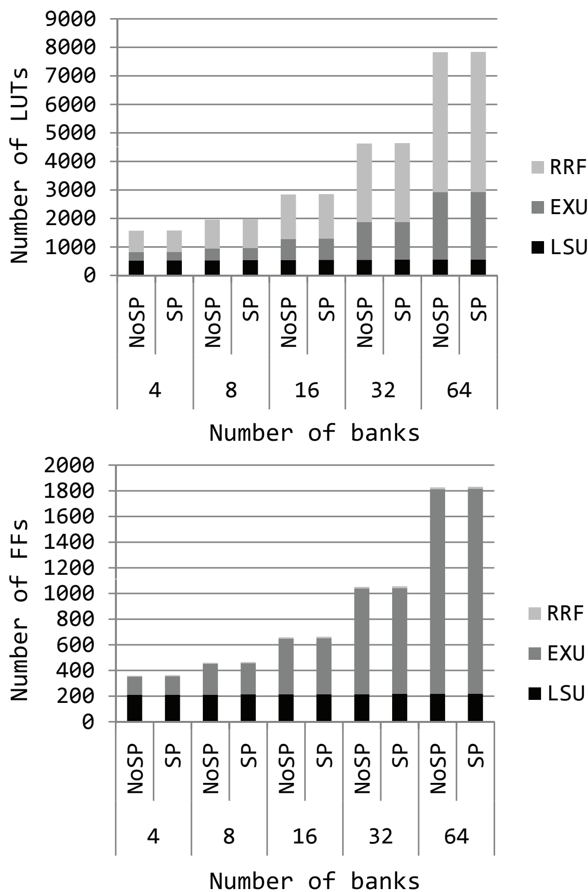

Figure 7: Implementation Result (Circuit Size).

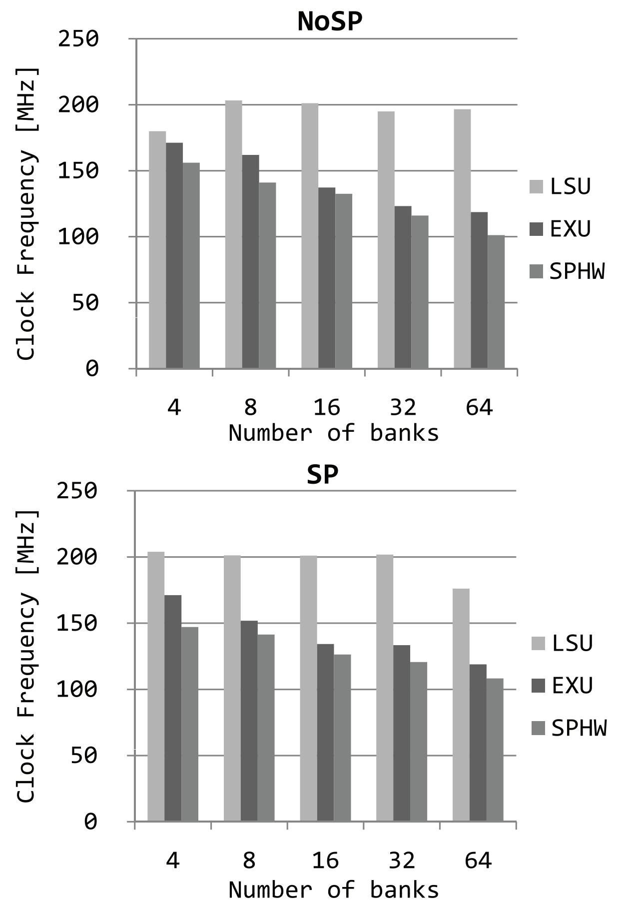

Figure 8: Implementation Result (Clock Freq.).

LSU is the fixed processor, the LUTs and FFs of it are constant.

Fig. 8 shows the clock frequency reported by the ISE12.1. The SPHW shows the worst clock frequency. This is because the critical pass resides on the pass from the DBRI to the multipliers in the EXU. The EXU hardware is generated by the handel-C. Thus, this critical pass is dependent on the C program to be compiled by the handel-C. The LSU does not affect the clock frequency.

Fig. 9 shows the performance result when the line transfer from the memory to the LSU consumes 6 clocks. The image size is 256×256. By hiding memory access latency, the SP can improve the performance of 1.67 to 1.72 times compared with the NoSP.

We were able to perform such tradeoff among the number of banks, the circuit size, the clock frequency, and the performance easily and quickly by only changing parameters.

6 Conclusion

The semi-programmable hardware is a design-level hardware architecture residing on the pass of which C program with memory accesses is converted to hardware. The SPHW realizes the memory access controller and the buffer by writing the software program and parameters respectively.

In this paper, we have introduced the SPHW as the data processing hardware into a real commercial HLS tool, Handel-C. By using the SPHW providing the register-based data access interface, we have demonstrated that the HLS tool can easily write the hardware accessing to the memory in C. This is because this interface hides the detail of the memory devices and the memory access patterns, by providing the data processing hardware with the simple stream data. For hiding memory access latency, the simple software-pipelining is able to be applyed to the memory access program and the parameters of the buffer. Consequently, we can realize the data processing hardware with data-prefetching mechanism at the complete C-level design entry, with lower burden.

As future work, we will introduce the SPHW into more HLS tools and evaluate using more application programs.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

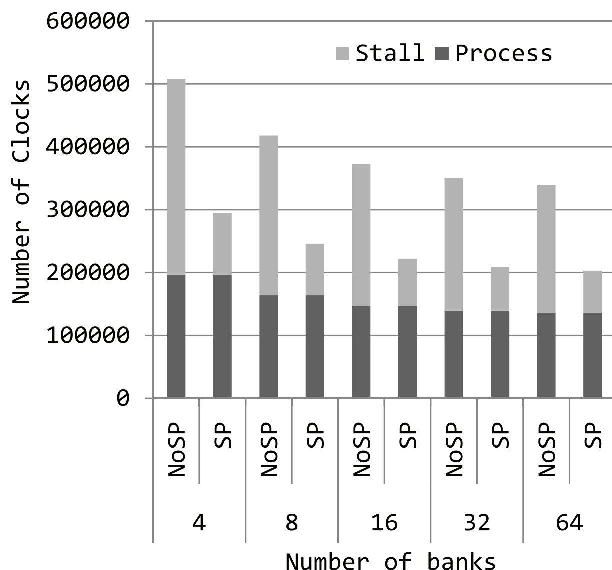

Figure 9: Performance Result.

References

[1] Mentor Graphics. Handel-c synthesis methodology. http://www.mentor.com/products/fpga/handel-c/, 2010.

[2] Sumit Gupta, Nick Savoiu, Nikil Dutt, Rajesh Gupta, and Alex Nicolau. Using global code motions to improve the quality of results for high-level synthesis. *IEEE Trans.on Computer-Aided Design of Integrated Circuits and Systems*, 23(2):302–312, 2004.

[3] David Lau, Orion Pritchard, and Philippe Molson. Automated Generation of Hardware Accelerators with Direct Memory Access from ANSI/ISO Standard C Functions. In *IEEE Symp. on Field-Programmable Custom Computing Machines*, pages 45–56, 2006.

[4] X. Liang and J. Jean. Data buffering and allocation in mapping generalized template matching on reconfigurable systems. *The Journal of Supercomputing*, 1(19):77–91, May 2001.

[5] Mitrionics. *Mitrion Users'Guide 1.5.0-001.* Mitrionics, 2008.

[6] J. Park and P. C. Diniz. Synthesis of pipelined memory access controllers for streamed data applications on FPGA-based computing engines. In *Proc. of intl. symp. on Systems synthesis*, pages 221–226, October 2001.

[7] D. Pellerin and S. Thibault. *Practical FPGA Programming in C.* Prentice Hall, 2005.

[8] S. P. Vanderwiel. Data Prefetch Mechanisms. *ACM Computing Surveys*, 32(2):174–199, 2000.

[9] Akira Yamawaki, Seiichi Serikawa, and Masahiko Iwane. An efficient hardware architecture from c program with memory access to hardware. In *Proc. of the 2010 International Conference on Computational Science and Its Applications, Part II*, pages 488–502, 2010.

[10] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)