Implementation and evaluation of AES/ADPCM on STP and FPGA with High-level Synthesis

Yuki Ando, Yukihito Ishida, Shinya Honda, Hiroaki Takada, Masato Edahiro
Nagoya University
Nagoya, Japan
{y_ando, ishider, honda, hiro, eda}@ertl.jp

Abstract— Reconfigurable techniques are attracting attention as an alternative to dedicated hardware of SoC. We have evaluated FPGA and STP engine in order to confirm their performance whether they can substitute the dedicated hardware of SoC. We selected AES and ADPCM applications to compare the performance of FPGA and STP engine. The applications were synthesized with the same high-level synthesis tools. Then, we implemented them onto FPGA and STP engine using the integrated development environments. For the evaluation, we compared them in terms of resource usage, the number of states, the number of cycles, frequency, and execution time.

I. Introduction

System-on-a-Chip (SoC) has been widely used inside mobile devices such as a cell phone and a digital camera, which realizes a processing of various data. A typical SoC consists of processors, standard input/output, memories, and a dedicated hardware for a specific application. Optimizations by software (SW) and hardware (HW) co-design have brought lower power consumption, smaller chip size, and high performances to the SoC. By consistent effort, the SoC is incredibly getting advance.

These days, because of the increase of processing data, functions which were previously implemented as SW in a system are desired to be implemented as dedicated HWs. This can be solved by adding dedicated HWs into the SoC. In this solution, however, a dedicated HW is required for each specific function, and various dedicated HWs are combined to realize a system. Conversion of function implementation from SW to HW brings larger chip size and higher power consumption. In addition, because the dedicated HW only supports a specific function, another dedicated HW is required in the SoC to realize a new function such as media encoding/decoding. On the other hand, because current semiconductor process takes longer time at last half of design such as place and route, costs to design a chip increase. Developing a new chip is carefully reviewed. For these reasons, a HW which is small in size, reconfigurable for functions, and fast in execution time is needed as a substitute of dedicated HW in the SoC.

An example of reconfigurable HW is a Field-Programmable Gate Array (FPGA). A FPGA is configured before its execution, and its configuration remains the same during its execution. Thus, a FPGA is a static and reconfigurable HW. FPGAs are getting advance as well as the SoC, and some of them are getting high performance as the same as the dedicated HW. In the FPGA, a basic element called Look-Up Table (LUT) is placed a lot. The reconfigurability of FPGA is realized by electrically changing the connection among LUTs (fine-grained reconfiguration). This technology brings the generation of various types of circuit on FPGA without waste of resources[1].

Another type of reconfigurable HW named "STP engine" is appeared and used in some products. This differs from FPGA in terms of reconfigurability. STP engine can dynamically change its configuration during its execution. Thus, STP engine is a dynamic and reconfigurable HW.

In this work, we implemented AES and ADPCM onto above two types of reconfigurable HW (FPGA, STP engine) and evaluated them. Although integrated development environments for FPGA and that of STP engine were used because the final target is different, same high-level synthesis tool was used for all implementations. A high-level synthesis is a technology which converts a behavioral description such as C language into a hardware description language (HDL) [2]. By changing the options of high-level synthesis tool, various HDL descriptions are generated from a single behavioral description. During the implementation, we used the same C description except the input/output description. In detail, the C description was converted into HDL description by a high-level synthesis tool, and then, the generated HDL description was converted to final implementation for each target. In other word, we focus comparison between architectures of reconfigurable HW. We compared the performance in terms of chip size, the number of states and cycles, frequency, and execution time. As the result, FPGA has shorter execution time than STP engine by reducing the number of states and cycles. On the other hand, STP engine is better than FPGA in terms of chip size.

This paper is organized as follows. Section II introduces a dynamic and reconfigurable HW named STP engine and Section III presents the implementation of

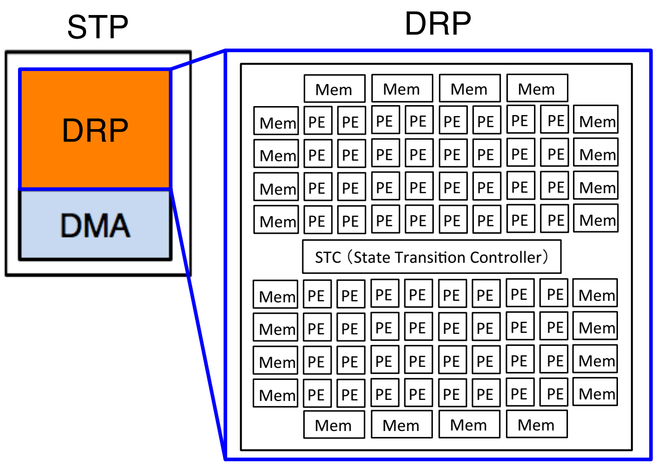


Fig. 1. Structure of STP

applications. In Section IV, we describe the evaluation of applications. Section V concludes the paper.

II. Dynamic reconfiguration and STP engine

This section shows a detail of STP engine and architecture of STP.

A. Dynamic reconfiguration

Technology called dynamic reconfiguration uses reconfigurable HW as well as FPGA. FPGA is only configured once before its execution. On the other hand, the other system such as STP dynamically changes its configuration during its execution. Thus, such system is called "dynamic reconfiguration". A basic element is called Processing Element (PE), which is more coarse-grained than LUT of FPGA and consists of combination of registers, multiplexors, and shifters (coarse-grained reconfiguration). This characteristic brings an advance in flexibility and a high performance for complex calculation processing. In addition, by reconfiguring its HW structure during the execution, its implementation can be larger than available resources. Therefore, resource efficiency gets better.

B. STP engine

STP (Stream Transposer) engine has a flexibility of SW and a high performance of HW [3]. STP can be implemented by software description written in C language through a high-level synthesis inside the development environment. A high load processing for a processor can be accelerated by STP. By using STP, a tolerant system of change can be realized.

STP consists of DRP and DMA as shown in Fig. 1. DRP is an array processor architecture which is dynamically reconfigurable. DRP is a core part of STP. In this work, we have implemented and evaluated several applications on the DRP. Several kinds of DRP exist, and the target of this work is DRP-2.

DRP consist of a finite state machine (FSM) and a data path, and its architecture efficiently realizes

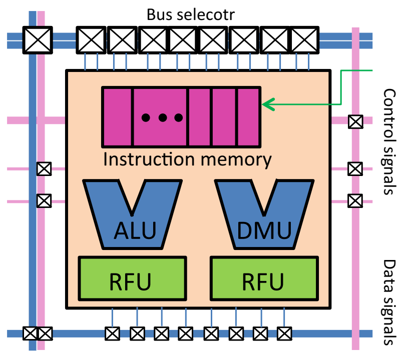


Fig. 2. Structure of PE

these two. The FSM is called STC (State Transition Controller) and controls the order of processing. Also, STC indicates processing contents by transmitting a "state" to the data path on each cycle.

The data path consists of arrays of PEs and memories. Fig. 2 shows the structure of PE. A PE consists of Arithmetic and Logic Unit (ALU), Data Manipulation Unit (DMU), and two Register File Units (RFUs) which keep 8 bits data. The ALU and the DMU handle 8 bits and 1 bit calculation. In addition, a number of programmable connections exist among PEs and memories. By flexibly connecting PEs and memories, the data path realizes various processing and can execute several calculations in parallel.

In this way, the processing content varies by changing the state of data path on each cycle. Therefore, STP is one of dynamically reconfigurable architectures.

C. Relation between state and surface in STP

Relation between a "state" and a "surface" is important for STP. Fig. 3 shows the relation between the state and the surface.

A surface is like an instruction of SW. The surface indicates the combination of PEs and connections of PEs. Multiple PEs can be used in a surface. Thus, a complex processing of calculation can be realized. By repeating the change of surfaces, the processing is executed.

A state is decided from a Data Flow Graph (DFG) by a scheduling process during the high-level synthesis. Although a state is basically allocated to a surface, states up to four can be allocated to a surface (multiple states in a surface). STP has up to 32 surfaces. Therefore, 128 states can be implemented on STP.

There are two methods to allocate multiple states in a surface.

- Type1: continuous states in a surface
- Type2: maximize the number of states in a surface

Type1 method allocates as many continuous states as possible in a surface under the resource constraints. This method is effective for reducing the number of change of surfaces. Type2 method allocates as many states as possible in a surface although the states are

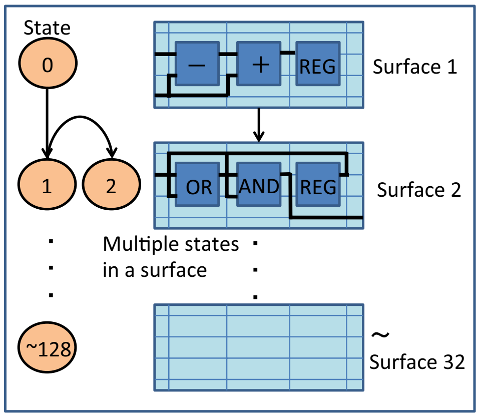


Fig. 3. Relationship between state and surface for STP

not continuous. This method maximizes the number of states in a surface, and is effective for reducing the number of surfaces. However, the overhead of changing the surfaces increases because the states in the surface are not continuous. Multiple states in a surface improve the resource utility. On the other hand, the overhead increases because of complex data path in a surface. Therefore, a trade-off of multiple states in a surface should be considered.

TABLE I
Brief description of the CHStone benchmark

Domain	Name	Description
Arithmetic	DFADD	Double-precision floating-point addition
	DFDIV	Double-precision floating-point division
	DFMUL	Double-precision floating-point multiplication
	DFSIN	*sine* function for double-precision floating-point numbers
Micro processor	MIPS	Simplified MIPS processor
Media processing	ADPCM	Adaptive differential pulse code modulation decoder and encoder
	GSM	Linear predictive coding analysis of global system for mobile communications
	JPEG	JPEG image decompression
	MOTION	Motion vector decoding of the MPEG-2
Security	AES	Advanced encryption standard
	BLOWFISH	Data encryption standard
	SHA	Secure hash algorithm

III. Implementation of applications

In this section, we describe the selection reason of applications that were implemented in the FPGA and the STP. Also, we describe how we implemented the applications.

A. *Selection of applications*

We selected applications that were implemented in the FPGA and the STP from CHStone benchmark [4]. CHStone benchmark is a suite of benchmark programs for high-level synthesis. In the past, benchmark programs for high-level synthesis were written in VHDL. CHStone benchmark differs from them because the programs in CHStone benchmark are written in C language. The programs in CHStone benchmark are comparatively complex and huge. In the program, descriptions that most of high-level synthesis tools does not support (e.g., recursive call of function, dynamic allocation of memory) are not used. In addition, the test vectors are included in the program and outside library is not required. For these reasons, we selected CHStone benchmark to implement and evaluate applications with a high-level synthesis tool.

Table I shows all programs in CHStone benchmark. Ideally, all programs should be implemented on the FPGA and the STP. However, STP has strict synthesis and resource constraint. Thus, we chose two applications. One is AES from security and the other is ADPCM from media processing. AES consists of an encryption and a decryption, and ADPCM consists of an encode and decode. In the STP, implementation of both encryption/encode and decryption/decode was impossible under strict resource constraint. For this reason, we separated AES into two applications, aes_enc and aes_dec. Aes_enc and aes_dec are an encryption and a decryption application, respectively. And we separated ADPCM into two applications, adpcm_enc and adpcm_dec, as well. These four applications can be implemented on the STP. As a reference value of implementation, we also implemented MIPS from microprocessor. Although the resource constraint of FPGA is more relaxed than STP, we implemented above five applications in the FPGA as well.

B. *Implementation on STP*

We used an integrated development environment named Musketeer to implement applications in the STP. To synthesize the implementation by Musketeer, C descriptions have to be converted to BDL description. BDL description is an extension of C language, which supports a description of HW. At least, following processing is needed.

- To change the name of header function
- To include specific library
- To add an input/output description

In this work, about 10 lines of code were added and a few lines of code were modified to convert a C description into a BDL description. After these modifications, the implementation of five applications was synthesized for the STP. Musketeer provides a synthesis report which indicates the number of states, the number of surfaces, and usage of resources. In

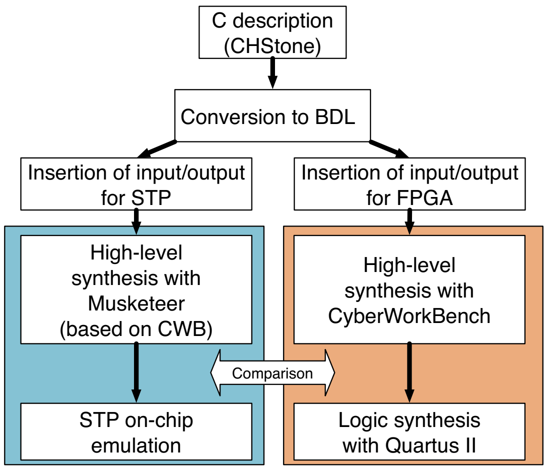

C description (CHStone)
Conversion to BDL
Insertion of input/output for STP
Insertion of input/output for FPGA
High-level synthesis with Musketeer (based on CWB)
High-level synthesis with CyberWorkBench
Comparison
STP on-chip emulation
Logic synthesis with Quartus II

Fig. 4. Flow of application implementation

addition, an on-chip emulation with generated configuration file provides maximum frequency and the number of execution cycles.

The implementations of aes_enc, aes_dec, adpcm_enc, and adpcm_dec have multiple states in a surface because the number of states is more than 32. On the other hand, the implementation of MIPS does not have multiple states in a surface.

C. Implementation on FPGA

We used an integrated development environment named CyberWorkBench [5] to implement applications in the FPGA. To synthesize the implementation by CyberWorkBench, C descriptions have to be converted to BDL description as well as Musketeer. Thus, we modified the C descriptions and converted them into BDL descriptions. Note that the input/output description for FPGA differs from that for STP. In addition, we minimized the modification to compare the FPGA and the STP equally.

After these modifications, the implementation of five applications was synthesized for the FPGA. CyberWorkBench provides a synthesis report which indicates the number of states. By simulating the implementation in cycle accuracy, the number of execution cycles was measured.

CyberWorkBench also outputs a HDL description of inputted BDL description. We used QuartusII which is a logic synthesis tool to measure the maximum frequency and resource usages. QuartusII took the generated HDL description as input and outputted a report of the maximum frequency and resource usages.

IV. Evaluation

A. Evaluation method

The developing environment is desired to be the same in order to compare the performances between FPGA and STP. Table II shows development environments. Fig. 4 shows a flow of application implementation in this work. The input of the flow is C descriptions in CHStone. The C descriptions were firstly converted to BDL descriptions. Then, an input/output mechanism for each target was inserted into BDL descriptions. The implementation tool is different for each target. Musketeer is used For STP while CyberWorkBench is used for FPGA. Because these two tools use the same high-level synthesis tool, common implementation environment is used till high-level synthesis.

TABLE II
Development environment

STP	High-level synthesis	Musketeer (Based on CyberWorkBench ver 5.0)
	HW	STP engine (DRP-2)
FPGA	High-level synthesis	CyberWorkBench 5.2.2.1 Enterprise
	Logic synthesis	QuartusII ver 8.1
	HW	StratixII (EP2S180F1020C3) CycloneII (EP2C70F672C6)

In terms of resource usage, a comparison between FPGA and STP is hard because their reconfigurable technology is different. To keep the equality of comparison, at least, semiconductor process should be the same. DRP-2 was developed by 90 nm process. We therefore chosen Altera's StratixII and CycloneII FPGA, which are developed by 90 nm process [6]. In addition, amount of resources are different between StratixII and CycloneII. We implemented applications on both StratixII and CycloneII to equally compare the performances with STP.

After the implementation of application by each development environment, the performance was evaluated in terms of resource usage, the number of state, the number of cycles, frequency, and execution time. On STP, Musketeer indicates those values except execution time after the implementation. On FPGA, after the conversion from C description to HDL by CyberWorkBench, QuartusII reports evaluation values except execution time. The execution time was calculated by equation (1).

$$exe\_time(\mu s) = \frac{\#cycle}{freqency(MHz)} \tag{1}$$

In the equation, exe_time and # cycle indicate execution time and the number of cycles, respectively.

The synthesis constraint of high-level synthesis tool effects the implementation. We implemented applications under frequency constraints of 30 MHz and 50MHz on both FPGA and STP. The optimization option of high-level synthesis tool also effects the implementation. We synthesized the implementations with two types of option below:

- Fast: To maximize execution speed
- MinUse: To maximize the execution speed under minimum resource usage

Common optimization method for FPGA and STP are shown below:

TABLE III
Summary of aes_enc implementation

target	FPGA (StratixII)		FPGA (CycloneII)		STP (DRP-2)	
option	Fast	MinUse	Fast	MinUse	Fast	MinUse
# ALUT	1,320	1,097	2,711	2,711	—	—
# Register	341	343	328	328	—	—
# DSP	0	0	0	0	—	—
# PE	—	—	—	—	27	26
# State	27	28	27	27	106	103
# Cycle	221	261	221	221	2,258	2,185
Freq. (MHz)	107.45	120.24	75.52	75.52	40	32
Exe. time (μs)	2.1	2.2	3.0	3.0	56.5	68.3

TABLE IV
Summary of aes_dec implementation

target	FPGA (StratixII)		FPGA (CycloneII)		STP (DRP-2)	
option	Fast	MinUse	Fast	MinUse	Fast	MinUse
# ALUT	2,814	2,604	6,415	6,415	—	—
# Register	1,484	1,458	1,455	1,455	—	—
# DSP	0	0	0	0	—	—
# PE	—	—	—	—	72	35
# State	42	44	44	44	93	93
# Cycle	788	864	864	864	3,085	3,085
Freq. (MHz)	98.14	97.76	81.75	81.75	41	39
Exe. time (μs)	8.1	8.9	10.6	10.6	75.3	79.1

TABLE V
Summary of adpcm_enc implementation

target	FPGA (StratixII)		FPGA (CycloneII)		STP (DRP-2)	
option	Fast	MinUse	Fast	MinUse	Fast	MinUse
# ALUT	4,197	4,535	9,172	8,513	—	—
# Register	3,193	1,496	2,658	1,897	—	—
# DSP	54	54	55	47	—	—
# PE	—	—	—	—	135	135
# State	44	123	102	121	80	80
# Cycle	1,992	2,869	1,872	3,085	7,996	7,996
Freq. (MHz)	110.57	120.61	64.40	50.19	26	26
Exe. time (μs)	18.1	23.8	29.0	61.5	307.6	307.6

TABLE VI
Summary of adpcm_dec implementation

target	FPGA (StratixII)		FPGA (CycloneII)		STP (DRP-2)	
option	Fast	MinUse	Fast	MinUse	Fast	MinUse
# ALUT	5,414	5,399	9,703	10,500	—	—
# Register	5,744	4,180	8,486	4,129	—	—
# DSP	84	72	57	49	—	—
# PE	—	—	—	—	210	181
# State	103	103	103	103	63	51
# Cycle	1,228	1,228	1,228	1,228	6,063	5,455
Freq. (MHz)	110.86	99.30	95.57	77.72	41	26
Exe. time (μs)	11.1	12.4	12.9	15.9	147.9	209.9

- **Speculative execution**
- **Loop unrolling**
- **Comprehensive parallelization of branch sentence**

B. *Implementation results*

Tables III, IV, V, VI, and VII show the summary of implementations for aes_enc, aes_dec, adpcm_enc, adpcm_dec, and mips, respectively. Each table indicates an option of high-level synthesis tool, # (the numbers of) ALUT, # register, # DSP, # PE, # state, # cycles, frequency (Freq.), and execution time (Exe. time).

In terms of execution time, the implementations on FPGA are much faster than those on STP. In detail, the implementations on FPGA are at least two times (mips) and up to 20 times (aes_enc) faster than those on STP. This is because the implementations on FPGA have high frequency and small number of cycles. For AES application, the number of states for FPGA is half to 1/4 of that for STP. For ADPCM application, the number of states for STP is less than that for FPGA. In terms of resource usage, as the number of ALUT, DSP and register in FPGA increases, the number of PE in STP also increases.

C. *Comparison between FPGA and STP*

The implementations of five applications are compared between the FPGA and the STP in terms of execution time and resource usage. For implementation on the FPGA , the values of CycloneII are used. Fig. 5 shows the comparison of execution time. In the figure, the fastest execution time of each application is shown. Fig. 6 shows a resource usage of implementations in Fig. 5.

A trend of the execution time and resource usage between the FPGA implementations and the STP implementations is similar. This might be because the same C description and high-level synthesis tool were used.

The execution time of implementations on FPGA was much faster than that on STP. This is because the STP has strict resource constraints. Even if the high-level synthesis tool parallelizes the processing of applications, STP cannot implement parallelized processing in parallel under its resource constraint. On

TABLE VII
SUMMARY OF MIPS IMPLEMENTATION

target	FPGA (StratixII)		FPGA (CycloneII)		STP (DRP-2)	
option	Fast	MinUse	Fast	MinUse	Fast	MinUse
# ALUT	1,890	1,808	3,958	3,958	—	—
# Register	1,629	1,362	1,362	1,362	—	—
# DSP	16	16	12	12	—	—
# PE	—	—	—	—	97	53
# State	25	23	23	23	18	23
# Cycle	2,912	2,912	2,912	2,912	4,033	7,170
Freq. (MHz)	84.26	71.05	52.91	52.91	37	34
Exe. time (μs)	34.6	41.2	55.2	55.2	109	210.9

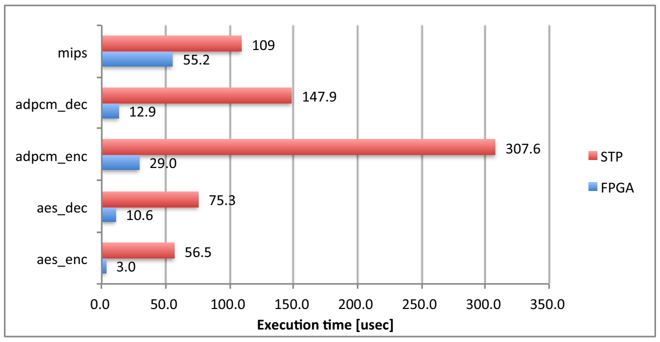


Fig. 5. Comparison of execution time

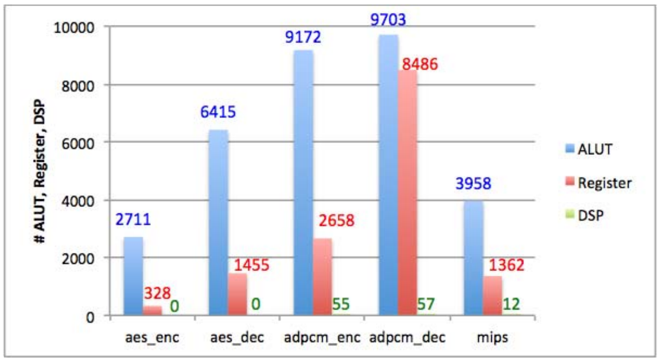
the other hand, the FPGA has enough resources to implement parallelized processing in parallel.

The direct comparison of resource usage is hard because architecture of the FPGA and STP is different. However, the rate of increase of resource usage has difference. Comparing the aes_enc and aes_dec of FPGA, the number of ALUT and the number of register increased about 2.5 times and about 4 times, respectively. Comparing the aes_enc and aes_dec of STP, the number of PE increased about 2.7 times. In addition, comparing the adpcm_enc and adpcm_dec of FPGA, the number of register increased about 3.2 times. Comparing the adpcm_enc and adpcm_dec of STP, the number of PE increased about 1.6 times. The results indicate that the STP can use resources more efficiently than FPGA.

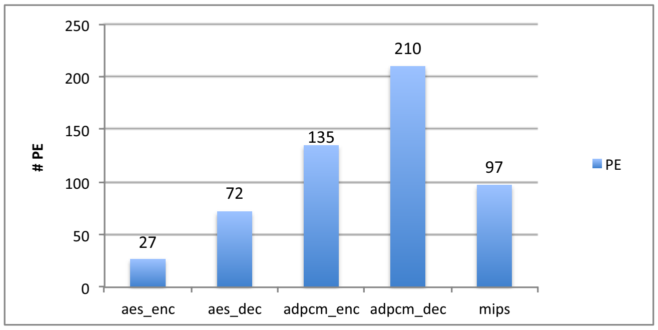
In this work, the implementations of FPGA are better than those of STP in terms of execution time. Meanwhile, the implementations of STP have high efficiency in resource usage. Therefore, STP has a possibility to be a substitute of dedicated HW in SoC.

V. Conclusion

In this work, we have evaluated FPGA and STP engine whether they can substitute the dedicated HW of SoC. We implemented AES and ADPCM applications using the same high-level synthesis tool to compare the performance of FPGA and STP engine. We compared the implementation of FPGA and STP in terms of execution time and resource usage. The execution time of implementation in FPGA was better than that of STP because the STP could not realize a highly parallelized implementation under strict resource constraint. Meanwhile, the implementations of STP have high efficiency in resource usage. Therefore, FPGA is suitable for high performance systems while STP can be a substitute of dedicated HW in SoC.



a) FPGA



b) STP

Fig. 6. Comparison of resource usage

References

[1] H. Amano, "A survey on dynamically reconfigurable processors," *IEICE TRANS. COMMUN.*, Vol. E89-B, NO. 12, pp. 3179-3187, 2006.

[2] D.D. Gajski, N.D. Dutt, A.C.-H. Wu, and S.Y.-L. Lin, "High-level Synthesis: Introduction to Chip and System Design," *Kluwer Academic Publishers*, 1992.

[3] Renesas Electronics Corporation, http://www.renesas.com/index.jsp

[4] Y. Hara, H. Takada, S. Honda, H. Tomiyama and H. Takada, "Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis," *Journal of Information Processing*, Vol. 17, pp. 242-254, 2009.

[5] K. Wakabayashi, "CyberWorkBench: integrated design environment based on C-based behavior synthesis and verification," *In Proc. of VLSI Design, Automation and Test, 2005*, pp. 173-176, 2005.

[6] Altera Corporation, http://www.altera.com/.