Multimedia IP Design in SystemC Design Environment Adopting High-level Code Optimization

Sik Kim, Kwang-Hyun Cho, Hyung-Jun Lim, Kyung-muk Lim, Eui-Young Chung,
Kyu-Myung Choi, Jeong-Taek Kong, Soo-Kwan Eo
CAE Center, System LSI Division, Semiconductor Business
Samsung Electronics

Abstract — *High-level design language such as SystemC has been gradually adopted to manage the increasing design complexity. In this paper, we describe the design experience of the NTSC/PAL video decoder in SystemC design environment. Especially, we focus on high-level code optimization techniques such as memory compaction and reusable communication interface design in addition to computing operation refinement to provide the source-level design controllability in behavioral design. Experimental results show that our method can reduce the design time from an algorithm to the gate-level netlist by more than 50% compared to the traditional RTL-based design method. Moreover, our method is competitive in terms of the design quality (area and latency) compared to conventional RTL-based method.*

Keywords: behavioral synthesis, code optimization, design time (TAT), SystemC

1 Introduction

Recent advances in nanometer process technology have extremely increased the device density and design complexity. According to the ITRS report in 2004, the estimated design size in 2015 will be 43.8 times larger than that of in 2004 meanwhile the design time will be reduced by 16.7% [1]. Therefore, it is strongly required to develop a design methodology that can increase design productivity to cope with the design complexity increase.

Algorithmic or behavioral synthesis introduced a higher level of design abstraction and is believed to reduce design time efficiently. However, the sequential nature of the algorithm SW require an additional description method to exploit parallel execution and increased difficulties in behavioral design description until SystemC [2], a C++ class library that can describe parallel execution and bit-wise data transfer, has been introduced.

In this paper, we describe a behavioral design procedure and behavioral-level code optimization techniques for NTSC/PAL video decoder application. Experimental results show that overall 50% design time reduction is possible when adopting our design method. We also measure the tool performance by comparing the design quality of the proposed behavioral synthesis flow and the traditional RTL flow.

This paper is organized as follows: Section 2 describes previous work related to the behavioral synthesis method and points out the differences in our work. Section 3 describes the target IP architecture of a Genlock processing block in the NTSC/PAL video decoder. Section 4 presents the overall design procedure in our design method and describes high-level code optimization techniques in behavioral design. In Section 5, we summarized the experimental results to compare the behavioral synthesis flow and the RTL flow in terms of design productivity. Finally, the conclusion and future works are described in Section 6.

2 Previous work

Many researchers have worked on improving the synthesis engine which results in better schedule, better binding, and so on [3][4]. Some other researchers have worked on enhancing its front-end to understand more various syntaxes and semantics such as pointer, struct, and so on [5]. On the contrary, our work is focusing on the code refinement and optimization method for the given synthesis environment. Code transformation or refinement has been widely researched in the embedded software society. They showed its effectiveness in many literatures [6][7]. Similar effort has been also done for HW implementation [8]. It shows the impact of memory access optimization using the ATOMIUM tool set. Even though their work shows remarkable results, its practical use is limited because their method is aiming at both software optimization and hardware implementation; thus, they have to tune the optimized code to be synthesizable for the chosen synthesizer. For this reason, the gain for design time was marginal. [9] shows interface synthesis techniques for efficient communication interface design. Their work aimed at handling various protocols; thus, the synthesized code generates more area overhead compared to a manual design.

Our work is different from these works in the sense that i) our method transforms the code while keeping synthesizability; thus, the code refinement for the synthesis is not necessary, ii) our communication interface design is targeting a single bus protocol; thus, well-customized SystemC template is designed for area efficiency. This template can be easily extended to handle other bus protocols.

3 Target IP architecture

The digital NTSC/PAL decoder provides many advantages, such as good video quality, minimum analog adjustment and ease of use. The NTSC/PAL decoder performs Genlock, YC separation, digital video adjustment, analog gain control and chrominance demodulation.
In Figure 1, we presented the Genlock processing block architecture of the NTSC/PAL video decoder. The purpose of the Genlock processing is to reconstruct a sample clock and the timing control signals such as horizontal sync, vertical sync and the color sub-carrier from the video signal. Since the original sample clock is not available in video decoder, it is usually generated by multiplying the horizontal line frequency by the desired number of samples per line, using a phase-locked loop [10]. However, there are several problems such as video noise which makes the determination of sync edges unreliable, mechanical limitation of analog VCR input causes slightly different timing interval, etc. Therefore, the Genlock processing is most complex part of the video decoder, to generate reliable signals along with lots of exceptional cases mentioned before. The functionality of sub modules in Genlock processing summarized as follows:

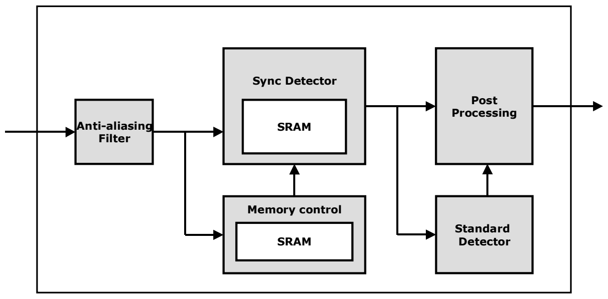

Figure 1. Genlock processing architecture.

4 Synthesis and code optimization

In Figure 2, we describe the overall design process to achieve efficient HW generation. The behavioral design flow has seamless refinement procedure compared to that of RTL flow. The last of this section focuses on the higher abstraction level design process, high-level code optimization, compared to the traditional design flow.

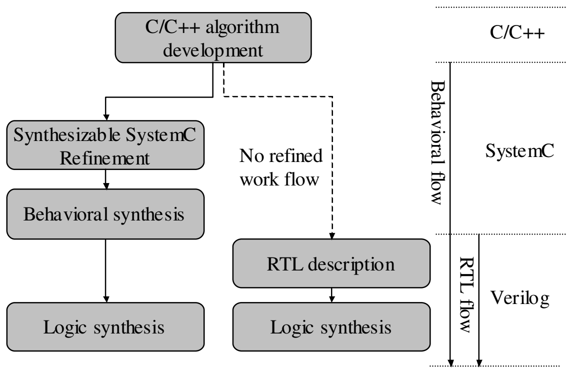

Figure 2. Design process comparison.

4.1 Functional module decomposition

Module decomposition results are summarized in Figure 3. The module decomposition process is mainly based on the functionality of the Genlock processing block and we tried to merge functional blocks if possible to reduce communication between modules that may results in additional registers to save temporal results.

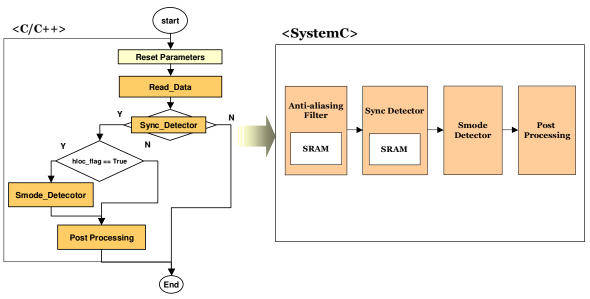

Figure 3. Module decomposition result for Genlock circuit.

4.2 Behavioral code refinements

In this section, we exploit several refinement steps that are required to get an optimal design output such as memory optimization, bit-width optimization and code translation.

- Memory optimization

Usually, algorithm SW adopts dynamic memory allocation for efficient memory usage in the target processor and uses the temporary copy of the memory to enhance readability. However, when we use the algorithm code as an input to the behavioral synthesis, we have to use static memory allocation so that it can be mapped on the real memory/register file. In addition, it is required to reduce the memory size itself because the number of storage elements has a direct impact on the design area.
In Figure 4, we present our refinement procedure to reduce the memory size in the Genlock processing block. We

performed lifetime analysis for each data element and find the minimum memory size manually. We confirmed our manual refinement results by applying ATOMIUM tool set to the same algorithm block [8]. It is promising that the ATOMIUM tool set can be exploited to automate the refinement procedure for arbitrary complex code set.

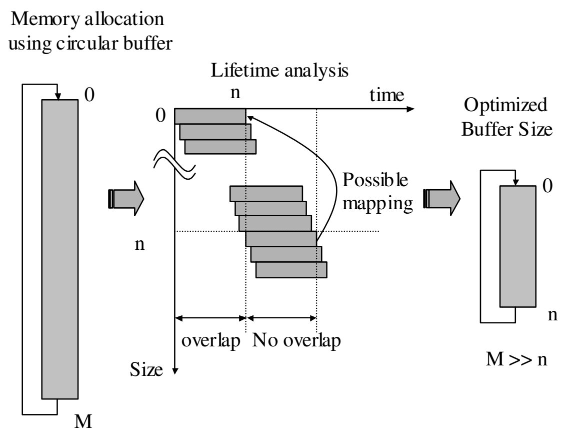

Figure 4. Memory optimization for Genlock circuit.

● Bit-width optimization

A lot of research has been performed on fixed-point bit width optimization because the degree of bit-width optimization directly impact on the area performance. In Figure 5, we presented the area of numerous operators vs. the bit input bit width.

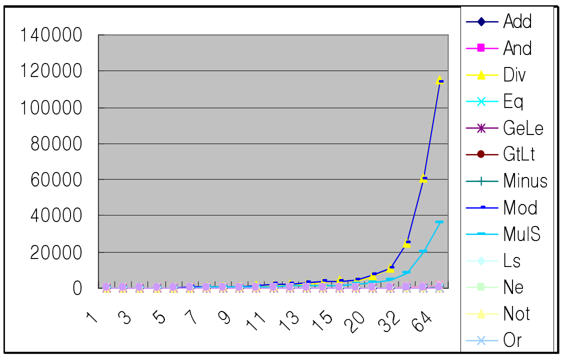

Figure 5. Area vs. bit width for basic operators.

One of the advantages in SystemC is that users can define the data size in an arbitrary number. Therefore, we are able to map the optimizing bit-width information directly.

● Operator refinement

Algorithm developers use complex mathematical functions such as sine function in their algorithm description and HW designers should translate the functionality by themselves. We found that it is possible to refine the SystemC code to describe a hardware behavior in detail. For example, we have found four 32 bit division operations in Feature Extractor and the division logic occupies over 50% of overall area when the operators are not shared during behavioral synthesis phase. Because only 5 LSBs of the division output are used, we performed the binary search rather than division itself. After the optimization, the overall area of the block is reduced to 74K gates. On the contrary, the area of a 32 bit division is 25K.

● Communication interface refinement

To integrate the IP in full chip environment, it is required to describe the AHB bus interface logic. We described the communication interface logic as a SystemC module and performed functional verification in an existing verification environment using Verilog, VERA and SystemC. In Figure 6, we show full chip IP verification system.

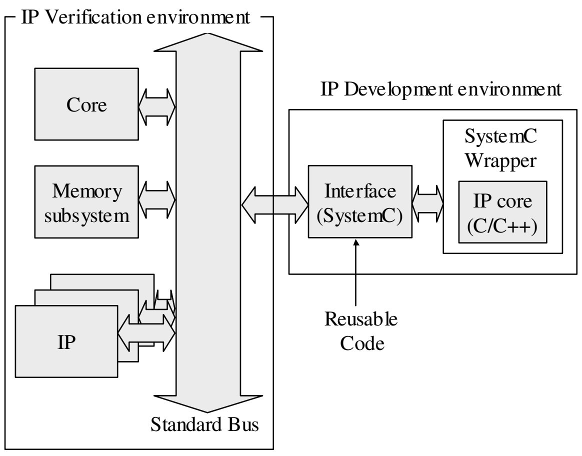

Figure 6. Communication interface design in the verification environment.

The communication interface FSM description in both the behavioral and the RTL model is nearly the same and it is clear that we cannot expect to shorten the description lines in control dominant block. However, it is possible to reuse the standard interface without much modification so that we can expect a similar speed-up in other applications.

5 Experiments

Table 1 presents the productivity information measured in our design experience using the Genlock block for video decoder.

Table 1. Productivity comparison.

Category	Unit	Results		Gain
		BEH	RTL	
Code length	# line	4,203	16,751	4X
Simulation time	sec	8	557	70X
Synthesis time	sec	1,893	16,056	8.5X
Area	# gate	238K	216K	0.9X
Design Cycle	M/M	3.25	6.9	2.1X

The table shows that the overall design time has been reduced more than 50%. We also present the code size, synthesis time, simulation time, design hierarchy information and architecture regeneration time to identify the factors that have contribution on design time reduction.
The overall behavioral code size was 50% smaller than the RTL code size in our design case. Data-path dominant block showed larger code size reduction when compared to that of control dominant block. For example, the behavioral description of data-path dominant filter has 800 lines while the RTL description of that module has 8K lines. In addition, in terms of simulation time, it is evident that performing simulation at the behavioral level can reduce the functional verification time.
In terms of HW architecture exploration, note that architecture exploration such as latency change can be easily managed in the behavioral design environment. In traditional RTL flow, FPGA prototyping of a real time constrained block often requires "re-design" of whole block to adjust block latency due to slow library characteristics and as a consequence using behavioral synthesis can save design and verification time in advance. After finishing the design procedure, we had experiments to regenerate the module with 1/2 latency constraints and the result was remarkable. The behavioral synthesis flow only took 50 minutes to regenerate module netlist while traditional flow required more than 1 week to re-write the RTL code.
We also performed behavioral synthesis on existing IP to evaluate the design quality of commercial design tools and summarized the logic synthesis results with each tool. To validate the capacity of behavioral synthesis tools, the target design size is very important and we selected 1 million gates IP which can be classified as very large block according to the ITRS roadmap [1]. Experimental results show that the RTL and the behavioral design are nearly equivalent in terms of gate count.

Table 2. Area performance comparison for behavioral tools and configurations.

IP	RTL flow	Behavioral flow			
		Tool 1		Tool 2	
		A	B	C	D
DFE	1.26M	1.31M	1.17M	1.29M	1.01M

6 Conclusion

The proposed paper has described the design experience using the SystemC-based IP design methodology. We have demonstrated that the overall design time can be reduced by 50% compared to that of traditional RTL design due to efficient description, easy architecture exploration and fast simulation at the behavioral level. The design optimization method for memory optimization and special HW description was provided to exploit more changes in behavioral design. We also examined the quality of commercial behavioral synthesis tools by comparing the design quality with existing IP developed in the traditional RTL design flow.
We expect that future research on sequential equivalence check between the behavioral model and the RTL model will increase the design productivity by avoiding RTL simulation to validate functional correctness.

References

1. ITRS reports on design, http://public.itrs.net/ , 2004

2. SystemC, OSCI, http://www.systemc.org/

3. R. Camposano, "Behavioral Synthesis," DAC 1996, pp. 33-34, June 1996.

4. N. When, et al., "Scehduling of Behavioral VHDL by Retiming Techniques", Proc. Of EuroDAC, 1994.

5. L. Semeria, and G. De Micheli, "Resolution, optimization and encoding of pointer variables for the behavioral synthesis from C", IEEE trans. On CAD, Vol. 20, Issue 2, pp.213-233, Feb. 2001

6. M. E. Wolf and M. S. Lam, "A loop transformation theory and an algorithm to maximize parallelism", IEEE trans. On Parallel and Distributed Systems, Vol. 2, Issue 4, pp. 452-471, Oct. 1991

7. M. E. Wolf, D. E. Maydan, and Ding-Kai Chen, "Combining loop transformations considering caches and scheduling", in Proc. of MICRO-29, pp. 274-286, Dec., 1996

8. F. Balasa et al., "Practical solutions for counting scalars and dependences in ATOMIUM-a memory management system for multidimensional signal processing", IEEE trans. On CAD, Vol. 16, Issue 2, pp. 133-145, Feb. 1997

9. J. Akella and K. McMillan, "Synthesizing Converters between Finite State Protocols", in Proc of ICCD'91, pp. 410-413. 1991

10. Keith Jack, Video Demystified 3rd edition, Eagle Rock VA, LLH Technology publishing, 2001