Implementation of a H.264 decoder with Template-based Communication Refinement

Sangyong Yoon, Sanggyu Park, and SooIk Chae
School of electrical engineering and computer science, Seoul National University
San 56-1 Sillim-dong, Gwanak-gu, Seoul, Korea
{syyoon, sanggyu, chae}@sdgroup.snu.ac.kr

Abstract— **We described an H.264 decoder implemented with our design methodology, in which a system function model of transaction level is first captured in SystemC and refined into RTL with a library of communication templates. We determined its communication architecture by exploring the design space with template-based communication refinement to meet its requirement of decoding VGA 30 frames per second at a clock frequency of 50MHz.**

Keywords—**H.264, template, communication, refinement**

I. INTRODUCTION

Several refinement-based design (RBD) methodologies [1][2] have been reported that can manage the complexity of the embedded system. In these methodologies, a system-level function is first captured at a higher abstraction level that does not include any architectural details and then converted to lower abstraction levels by progressively making architectural decisions. In a SystemC-based RBD methodology proposed by Grotker et al. [6], a system-level functional model is described with concurrent computation models of the behavioral level in C/C++ and their communications with channels.

In our design flow, a SystemC-based RBD methodology provides the Communication Architecture Template Tree (CATtree) library, with which the designers capture the communication function and refine the communication architecture. As described in [4], a CATtree for a communication primitive channel is a collection of communication architecture templates (CAT) which are parameterized implementations of the channel. Each CAT has a transaction level model (TLM) in SystemC for function-level simulation, a RTL model in VHDL for hardware implementation and a software template for software implementation.

An approach to refine communication architecture in [5] is similar to our approach. While they provides only FIFO channels and bus-based ones as a communication primitive, our CATtree library includes various communication primitives so that it we can model the video systems effectively.

In this paper, we describe implementation of the H.264 decoders using our design environment and mainly explain their communication architecture refinements with a CATtree library. In this paper, we do not include the architectural details of computation blocks because this paper focuses on the communication architecture refinement.

The rest of this paper is organized as follows. We explain our system design flow in Section 2 and then describe the CATtree library in Section 3. The communication refinement for the H.264 design examples is summarized in Section 4, which is followed by the conclusion.

II. DESIGN FLOW

The design flow we used in designing H.264 decoders consists of two steps: function modeling and architecture refinement, as shown in Figure 1. In the function modeling step, the designer captures a system-level function for H.264 decoders at the transaction level using SystemC (Figure 1a), which consists of computation TLMs manually captured in SystemC, and channel TLMs from the CATtree library.

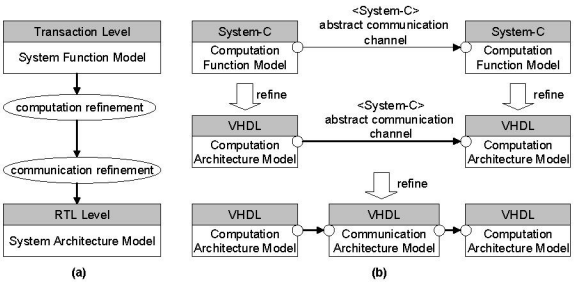

Figure 1. (a) System design flow and (b) Example of design entities.

In the refinement step, we refined computation models first. Each computation TLM can be refined to a RTL model in HDL manually or with a C-to-RTL synthesis tool. Note that we used computation models manually described in RTL because our design target was performance critical.

In the communication refinement step, for each channel in the design, we selected a proper architecture template from the CATtree library and replaced it with the original channel after configuring its parameters to meet the design constraint. We verified functional correctness of the refined system model with transaction-level simulation. In refining some part of the system into RTL, we checked the function and performance in RTL with our mixed-level simulation environment [3].

III. COMMUNICATION LIBRARY

A CATtree (Communication Architecture Template Tree), which is defined for each communication primitive, is used in

1-4244-0387-1/06/$20.00 ©2006 IEEE

system function modeling and communication architecture refinement [4]. It includes its function-level model in TLM and various communication architecture templates (CATs) with different performance and area characteristics. For functional verification, a SystemC-based transaction-level model is provided for each channel. And for implementation, a RTL model and a software model are provided for each CAT.

Figure 2 shows a one-dimensional array (1DArray) channel, which is an abstraction of addressable memories. The 1Darray channels include Register channel, which is based on flip-flop implementation, SSRAM channel, which consists of an on-chip SSRAM controller and an on-chip SSRAM, and Cached channel, which contains local caches.

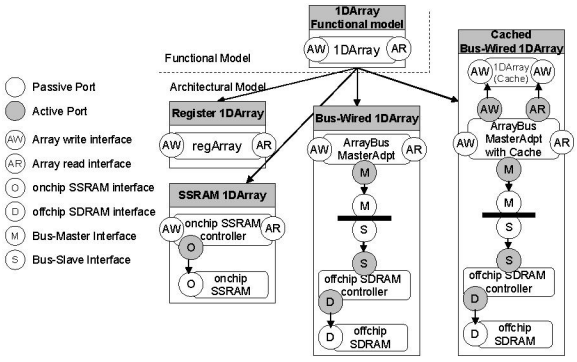

Figure 2. A CATtree for 1D array channels.

A. *Function-Level Models*

The CAT library in our design environment supports several types of the channels such as FIFO, array, event and variable channels.

A FIFO channel is for point-to-point, ordered and synchronized data transmission, which are suitable to model the channels between two computation blocks if they have data-dependency. An event channel is for point-to-point event notification without data transmission, which is useful to model the channels between two computation blocks if they have control-dependency. An array channel is good for the channels with data storage that are addressable with index, which does not include any synchronization function. We support three kinds of array channels such as 1D, 2D, and 3D arrays. A variable channel is useful to model the channels with data storage with multiple writers and readers, which does not have any synchronization function either.

B. *Architecture-Level Models*

A CATtree is provided for each communication primitive and its root includes a SystemC TLM model as its function-level model, which does not have any architecture details. Therefore, during the communication refinement step, we refine it to a specific architecture of each channel by selecting one of the CATs in its corresponding CATtree to meet the design constraints for a specific application. The CATs may include the following parameters.

(1) Buffer's memory size : FIFO depth or array size
(2) Buffer's memory type : register, on-chip memory, or off-chip memory
(3) Cache configuration : cache size, and cache line size
(4) Bus topology : point-to-point, or shared bus
(5) Bus arbitration scheme : priority, round robin

If we change an architecture parameter in a CAT, the variation of its performance and area can be significant. We modeled a decoded frame buffer of H.264 decoder with a 3D array channel, which includes an off-chip SDRAM as its buffer memory. To find a good architecture, we changed the size of cache embedded in the array channel as well as its cache line size and then measured the average number of clock cycles required to transfer the data per a macroblock processing from a decoded frame buffer to Inter Prediction block. As shown in Figure 3, the performance is changed from 787 cycles to 2667 cycles and the logic gate count also varies from 14.7K to 70.3K. In this experiment, we used the foreman bit-stream of seven QCIF pictures.

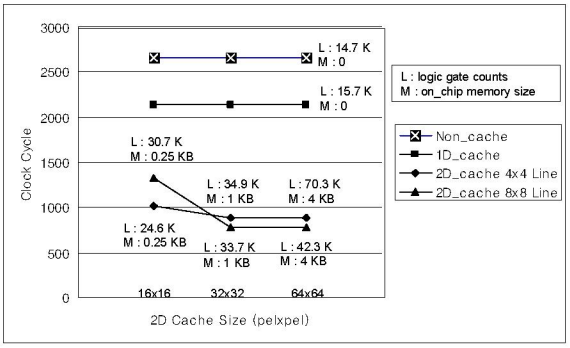

Figure 3. Simulation results for a 3D array by changing the parameters of the cache embedded

IV. COMMUNICATION REFINEMENT FOR H.264 DE-CODERS

We designed two different H.264 baseline decoders: one for VGA 30 frame/s and the other for CIF 30 frame/s at a clock frequency of 50 MHz. To have a margin of about 300 cycles, the decoders were targeted to decode one macro block in 1100 cycles for VGA pictures and in 3900 cycles for CIF pictures, respectively. Similarly, their critical path delays were limited to be less than 15ns.

As shown in Figure 4, we first captured a system function model, which is a TLM in SystemC, for the H.264 decoders. It consists of seven computation blocks: a bitstream parser (PARSER), a variable length decoder (VLD), an inverse transform and inverse quantization (ITQ) block, an inter-picture prediction (INTRA) block, an intra-picture prediction (INTER) block, a reconstruction (RECONST) block, and a de-blocking filer (DF). Note that bit-stream parser controls all the computation blocks, which is not shown in Figure 4 for the clarity. Basically each computation blocks are pipelined in MB level, but INTER, ITQ, INTRA and RECONST are pipelined in sub-MB level to reduce the channel's buffer size.

To model the communication among the computation blocks, we used 65 FIFO channels, 18 array channels, and 3

variable channels. Note that only the important channels in the H.264 decoder are shown in Figure 4.

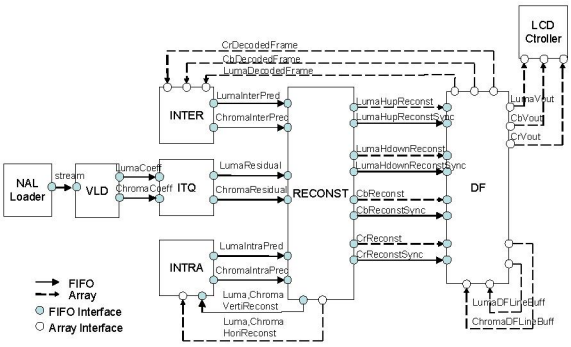

Figure 4. System function model of the H.264 decoder.

After capturing the function model of the H.264 decoder, we manually refined each computation block into RTL. Their performance and complexity are summarized in Table 1.

TABLE I. DESIGN SUMMARY FOR THE COMPUTATION BLOCKS.

Computation Block	Area (K gates)	Performance (cycles / macro block)
VLD	4.7	444
ITQ	15.9	150
INTRA	24.6	478
INTER	47.1	507
RECONST	2.0	260
DF	76.4	503
Total	170.7	

In designing the VGA decoder, we started with an initial configuration and went through five major refinement steps to get the final communication architecture, as shown in Figure 5. In order to evaluate the system area, we synthesized the decoder in 0.18 μm process technology with a 15ns timing constraint. To measure its system performance, we simulated the whole H.264 decoder with a VHDL simulator while only PARSER was executed in software. In this experiment, we used the foreman bit-stream of forty QCIF pictures where QP is 28 and the maximal reference frame number is 15.

For the initial communication architecture, the most of FIFO channels were refined to register-implemented FIFOs [4] with the depth enabling MB level pipelining. Decoded frame buffers were refined to a non-cached 3D array that uses an off-chip SDRAM. Line buffers were refined to a SSRAM 1D array that uses an on-chip static SRAM. The other array channels were refined to a registered 1D array. This initial architecture implies that just like other conventional video decoders, we used an off-chip memory for the frame buffers, on-chip memories for line buffers, but for the other channels we tried to minimize their complexity.

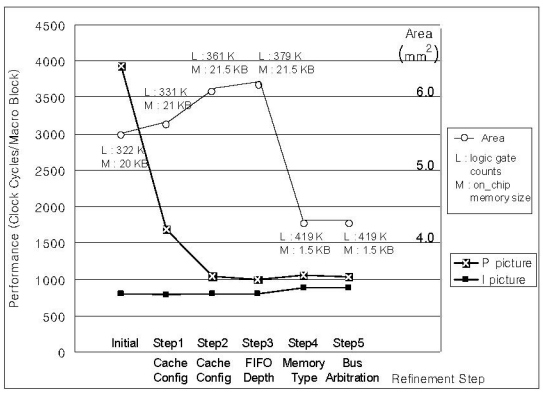

Figure 5. Performance and area of the H.264 VGA decoder after each refinement step.

After throughput analysis, we found that the array channel transferring the data from the decoded frame buffer to INTER block was a throughput bottleneck in the initial architecture because the INTER block wasted most of the cycles in waiting to read data from the decoded frame buffer. Therefore, we decided to use an array channel with a cache to reduce the latency of the array channel using off-chip SDRAM. At the refinement step 1, we configured the 3D array to one with a 2D cache of size 32x32 with line size 8x8, based on the results in Figure 3. In the refinement step 2, we configured two 3D arrays for the decoded frame buffers of chrominance to ones with a 2D cache of size 16x16 with line size 4x4. After the refinement steps 1 and 2, the performance was enhanced by four times with complexity overhead of 39 Kgates and 1.5KB on-chip memory.

After the step 2 refinement, we found that chrominance sample processing in the INTER block waited until luminance sample processing was finished in each macro block because luminance and chrominance processing shares the same datapath in the DF block. We resolved this problem by increasing the FIFO depth from 16 bytes to 256 bytes. Consequently, the performance was improved from 1049 cycles to 992 cycles by decreasing 57 cycles while the area was increased by 18 Kgates.

Initially the total size of the on-chip memory for the line buffers was 20 KB. Therefore, instead of the 1D array channels with on-chip memory for the line buffers, we also decided to use 1D array channels using the off-chip SDRAM in the refinement step 4. Consequently, with the performance penalty of 65 cycles, the gate counts were increased by 40 Kgates and the on-chip memory size was reduced by 20 KB, which is equivalent to roughly 240 Kgates. Consequently, its reduction of the silicon area was 1.9 mm^2, which is about 44% of the total silicon area of the final VGA decoder.

Because the frame buffers and the line buffers were decided to use the same SDRAM and the same bus in this design, its performance depended on the bus arbitration scheme. The initial bus arbitration scheme was round robin. Therefore, in the refinement step 5, we configured that the priority of LumaDecodedFrame channel was the highest and the other

channels were scheduled by a round robin policy, which enhanced the performance by 30 cycles without any penalty.

After all the five refinement steps, the performance was improved by about 4 times and the area was increased to 1.3 times and the on-chip memory size was reduced from 20KB to 1.5 KB. Consequently, we could meet the design specification of decoding VGA 30 frames per second at 37 MHz.

We also designed another H.264 decoder for CIF pictures with similar refinement steps from the same initial communication architecture. Both results are summarized in Table 2. In the CIF decoder, we decided not to use cache in the frame buffers because the performance constraint is looser than that of VGA decoder. We could reduce the system area by 0.7 mm^2 while meeting the performance constraint of CIF decoder.

TABLE II. DESIGN RESULTS OF VGA AND CIF DECODERS

Design Constraint	System Area			System Performance (cycles / macro block)	
	Logic (K gates)	Memory (KB)	Area (mm^2)	P Pic	I Pic
VGA 30 fr/s @ 50MHz	419	1.5	4.3	1027	887
CIF 30 fr/s @ 50MHz	369	0	3.6	3914	887

Although we designed the communication architecture by refining the templates in the CATtree library rather than by designing it manually, Table 3 shows the design results have reasonable system area and performance compared with other designs.

TABLE III. DESIGN COMPARISON WITH OTHER H.264 DECODERS

	[7]	[8]	Our Design
Profile	Baseline	Main	Baseline
Software Part	Parser+VLD	Non	Parser except VLD
Picture Size	HD	CIF	VGA
Clock Frequency	200 MHz	30 MHz	37 MHz
Logic Gates	300K	189K	419K
Onchip Memory	74 KB	41.5 KB	1.5 KB

V. CONCLUSION

In this paper, we explained a communication library, which supports various channel primitives and includes a SystemC TLM and various RTL architecture templates for each channel primitive, to support efficient communication architecture refinement. Since each channel template is a parameterized architecture model, the library is very useful for effective communication DSE.

We designed two different H.264 decoders with the CAT library. We found that our design flow was very efficient because the system function model capture for a VGA decoder can be reused directly in designing a CIF decoder. Furthermore, the communication DSE could be performed effectively with the CAT library we developed. The VGA decoder occupies an area of 4.3 mm^2 and can decode a macro block in 1027 cycles, while the CIF decoder occupies 3.6 mm^2 and in 3914 cycles.

REFERENCES

[1] J. Peng, S. Abdi and D. Gajski, "Automatic Model Refinement for Fast Architecture Exploration", Proc. ASP-DAC 2002, pp. 332-337, Jan. 2002.

[2] T. Givargis, F. Vahid, J. Henkel, "System-Level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip", IEEE Trans. on Very Large Scale Integration Systems, vol. 10, issue 4, pp. 416-422, Aug. 2002.

[3] Sanggyu Park, Sangyong Yoon, Soo-Ik Chae, "A mixed-level virtual prototyping environment for refinement-based design methodology", Accepted by Workshop of Rapid System Prototyping, Jun. 2006.

[4] S. Park, S. Chae, "Reusable component IP design using refinement-based design environment", Proc. of 11th ASP-DAC, Jan. 2006.

[5] J.-Y. Brunel, W. Kruijtzer, H. Kenter, F. Petrot, and L. Pasquier, "COSY Communication IP's", Proc. Design Automation Conf., pp. 406-409, June 2000.

[6] Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, "System design with SystemC", Kluwer Academic Publisher, 2002.

[7] Y, Hu, A. Simpson, K. MCAdoo, J. Cush, "A high definition H.264/AVC hardware video decoder core for multimedia SOC", IEEE International Symposium on Consumer Electronics, pp. 385-389, Sept. 2004

[8] Cheng-Ru Chang et al. "An H.264/AVC Main Profile Hardwired Decoder", 2006 Picture Coding Symposium, Beijing, China, April 24-26, 2006