PAPER

VLSI Implementation of a VC-1 Main Profile Decoder for HD Video Applications

Jinhyun CHO[†,††a)], *Student Member*, Doowon LEE[†††], *Nonmember*, Sangyong YOON[††], Sanggyu PARK[†], *Student Members*, *and* Soo-Ik CHAE[†], *Nonmember*

SUMMARY In this paper, we present a high-performance VC-1 main-profile decoder for high-definition (HD) video applications, which can decode HD 720p video streams with 30 fps at 80 MHz. We implemented the decoder with a one-poly eight-metal 0.13 μm CMOS process, which contains about 261,900 logic gates and on-chip memories of 13.9 KB SRAM and 13.1 KB ROM and occupies an area of about 5.1 mm^2. In designing the VC-1 decoder, we used a template-based SoC design flow, with which we performed the design space exploration of the decoder by trying various configurations of communication channels. Moreover, we also describe architectures of the computation blocks optimized to satisfy the requirements of VC-1 HD applications.

key words: *SMPTE 421M-2006 VC-1, video decoder, transaction level modeling, design space exploration*

1. Introduction

The Society of Motion Picture and Television Engineers (SMPTE) proposed a new video standard called VC-1 in March 2006 [1], which was derived from Microsoft WMV9. Recently both HD DVD and Blu-ray Disc have adopted VC-1 as one of their mandatory video standards. Moreover, Windows Vista includes a VC-1 decoder and its related components for HD DVD playback of VC-1. Although the VC-1 standard is a relatively new standard, it is expected to be as important as the H.264 video standard [2].

Because the VC-1 standard employs more software-friendly algorithms to obtain higher compression, the control path of the VC-1 decoder is known to be relatively more complex than that of the H.264 decoder in hardware implementation. A dataflow in the VC-1 decoding process is shown in Fig. 1.

The VC-1 standard has several distinct features. It adopts a DCT-like integer transform with four different block sizes such as 8×8, 8×4, 4×8, and 4×4. It supports AC/DC prediction for intra prediction, and bicubic and bilinear filtering for motion compensation. Pixel range reduction and intensity compensation can be optionally performed before motion compensation. Finally, both overlap smoothing and deblocking filters are employed to

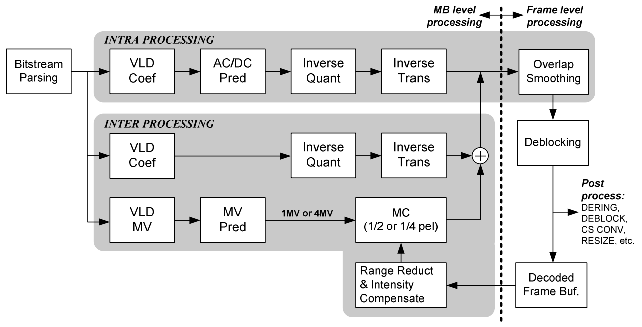


Fig. 1 Dataflow in the VC-1 decoding process.

reduce blocking artifacts.

In designing the VC-1 decoder, we used a template-based SoC design flow. It improves design productivity by exploiting its communication-channel template library which enables us to evaluate various communication architectures in order to find a better design. By using the efficient SoC design flow, we implemented the VC-1 HD 720p real-time decoder into silicon. To satisfy high memory bandwidth requirement of the decoder, we employed memory servers that include a cache, exploit bank-interleaving and prefetching, and separate two clock domains with asynchronous FIFOs.

In this paper, we explain the architecture of the components optimized for VC-1 HD 720p main profile decoding. For syntax parsing, we used a reordered merged VLD ROM table to reduce the number of ROM accesses. For inverse transform, we adopted architecture with butterfly operations to reduce the number of multiplications and additions. Moreover, we also optimized the datapath of interpolation filters for motion compensation to share it for several modes such as bicubic and bilinear filtering. For macroblock-level overlap smoothing and deblocking, we introduced a data-hazard free sequence of filtering and scrambled all the 4×4 sub-blocks into 8 single-port SRAM buffers for stall-free sub-block-level pipelining.

The rest of this paper is organized as follows. In Sect. 2, we explain a template-based SoC design flow employed in designing the VC-1 decoder. We explain system-level architecture of the decoder in Sect. 3 and describe its component-level architectures in Sect. 4. Then, we summarize its implementation results in Sect. 5. Finally, we will draw a conclusion in Sect. 6.

Manuscript received June 6, 2008.

[†]The authors are with the School of Electrical Engineering and Computer Sciences/Electrical Engineering, Seoul National University, 151-742 Korea.

[††]The authors are with the Mobile Next-generation Technology Team, Samsung Electronics Ltd., 446-711 Korea.

[†††]The author is with the Naval Academy of Korea, 645-797 Korea.

a) E-mail: holybit@sdgroup.snu.ac.kr

DOI: 10.1587/transfun.E92.A.279

2. SoC Design Environment

We briefly introduce a template-based SoC design environment, called SoCBase-DE [3], [4], which we used in designing the VC-1 decoder. SoCBase-DE, developed by Seoul National University, is an integrated environment for both design and verification. It provides a channel template library that includes model generators for many channel architecture templates (CATs) for efficient communication refinement, which can be regarded as a natural extension of the ASIC standard-cell library to a higher level.

2.1 SoCBase-DE Design Flow

Designers capture a transactional level model (TLM) of a system by using SystemC [5] after analyzing its algorithmic model in the SoCBase-De design flow, as shown in Fig. 2. In the TLM step, a system is separated into computation blocks and communication channels. Computation blocks are connected through channels, with which the system's communication network is composed. In the implementation step, the computation blocks are partitioned into software and hardware. In the SoCBase-DE, each hardware computation block is manually refined to its RTL while each software computation block reuses its SystemC TLM code as software code.

The SoCBase-DE provides the channel template library which supports four types of abstract channels such as FIFOs, variables, broadcasts, and arrays. In other words, the template library includes various CAT generators for each abstract channel. In the communication refinement step, we can generate various CATs by selecting a CAT generator for a specific micro-architecture in the channel template library and configuring its parameters [6]. Moreover, all necessary software interface codes for the refined channels are automatically generated.

Because a CAT is represented with three models such as a TLM model, a RTL model, and a software model, we

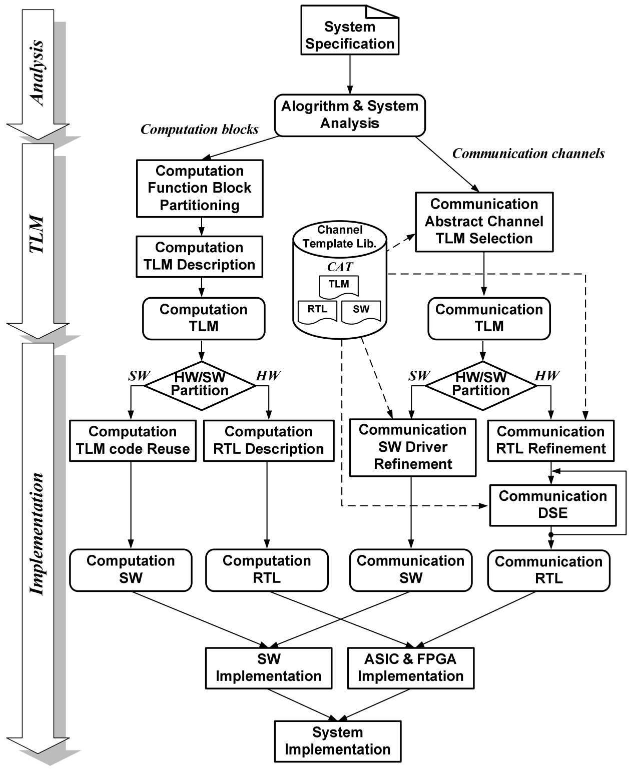


Fig. 2 SoCBase-DE design flow.

```
#--------------------------------------------------------------------
# SoCBase-DE AST (Architecture Synthesis Tool) Script
#--------------------------------------------------------------------
# # -------------- (I) : Before CAT refinement ----------------
# -------------- Computation Block Interfaces ----------------
# VC-1 LF Computation
computation vc1_LF_cmpt
   interface var_write <u_int8> o_var;
   interface array_write <u_int32,1024> o_arr_to_RL;
end;
# VC-1 Ref.Loader Computation
computation vc1_RL_cmpt
   interface var_read <u_int8> i_var_a;
   interface array_read <u_int32,1024> i_arr_fr_LF;
   interface FIFO_put <u_int12> o_FIFO_to_MC;
end;
# VC-1 MC Computation
computation vc1_MC_cmpt
   interface var_read <u_int8> i_var_b;
   interface FIFO_get <u_int12> i_FIFO_fr_RL;
   interface array_write <u_int32,1024> o_arr_to_MC;
   interface array_read <u_int32,1024> i_arr_fr_MC;
end;
# ------------------ Top Design and Connections -----------------
# Top
design vc1_dec
   # instances
   instance vc1_LF_cmpt  uLF_cmpt;
   instance vc1_RL_cmpt  uRL_cmpt;
   instance vc1_MC_cmpt  uMC_cmpt;
   # connection
   connect uLF_cmpt.o_var  uRL_cmpt.i_var_a  var_V;
   connect uLF_cmpt.o_var  uMC_cmpt.i_var_b  var_V;
   connect uRL_cmpt.o_FIFO_to_MC
              uMC_cmpt.i_FIFO_fr_RL  FIFO_F;
   connect uLF_cmpt.o_arr_to_RL  uRL_cmpt.i_arr_fr_LF  arr_A;
   connect uMC_cmpt.o_arr_to_MC  uMC_cmpt.i_arr_fr_MC  arr_B;
end;
# -------------- (II) : CAT refinement (DSE) ----------------
# A array refinement to SDRAM array
select arr_A;
refine selected to ext_sdram_bus_array;
unselect -all;
# B array refinement to on-chip SRAM array
select arr_B;
refine selected to onchip_ssram_array;
unselect -all;
# FIFO depth refinement
select FIFO_F;
refine selected finite_FIFO <-,20>; # FIFO <width, depth>
unselect -all;
# ------------------ Template-Code Generations ----------------
# RTL/SystemC template generation
write –verilog uLF_cmpt; //-sc (SystemC TLM), -verilog (RTL)
write –verilog uRL_cmpt;
write –verilog uMC_cmpt;
write –all; //CAT RTL, TLM, SW code generation
```

Fig. 3 Design script for a channel refinement example.

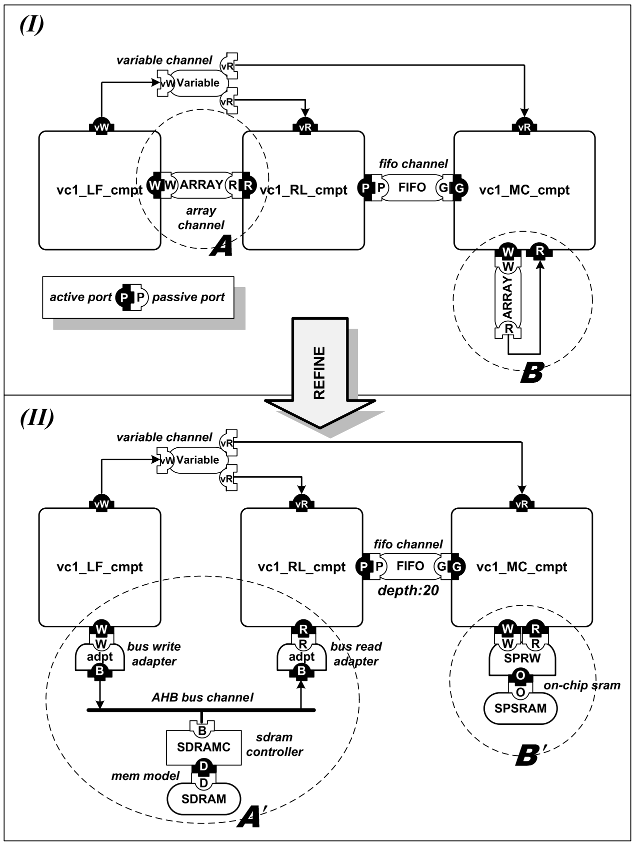


Fig. 4 An example of channel refinements.

can refine it into hardware or software by selecting one of its corresponding CAT models. Because all their interfaces to the computation blocks in each abstract level are predefined, we can easily refine them without modifying their interfaces.

2.2 Refinement of Communication Channels

Unlike the platform-based design methodology (PBD) [7], SoCBase-DE allows us to start with a flexible communication network and refine it to various configurations for channels and memory architectures. As an example of communication channel refinement for the SoCBase-DE design flow, a design script and its corresponding block diagram are shown in Figs. 3 and 4, respectively. An array channel A is refined to a SDRAM-based array channel (A′) with AHB bus; an array channel B to an on-chip SRAM based array channel (B′).

3. System-Level Architecture

After analyzing the VC-1 algorithm and its reference software, we designed the VC-1 decoder by dividing it into eleven components as shown in Fig. 5. Just for efficient explanation, we partitioned them into three major parts: syntax parsing, image reconstruction, and loop filtering. Before implementing the components, we first captured a system-level TLM using the abstract channels from the CAT library, as shown in Fig. 6. FIFOs are used for a low volume of data transfers with synchronization; on the other hand, arrays are used for a high volume of data transfers.

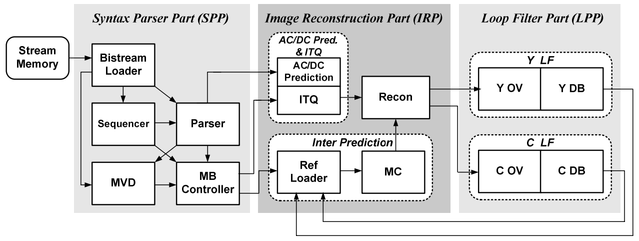


Fig. 5 Block diagram of the VC-1 video decoder.

In the communication DSE step, we can refine array channels to either an external SDRAM/DDR2 memory system or an internal on-chip SRAM memory system. To satisfy the requirements for the HD-level performance and area, we refined the system TLM by refining its channels as shown in Fig. 7. According to the profile data of the VC-1 decoder [8], the sequence-level parsing part occupies less than 5%, which is relatively less complex than other function blocks. Therefore, we implemented it in software for flexibility while all the other computation blocks are implemented in hardware for high performance.

It is necessary to provide high memory bandwidth in designing an HD-level video decoder. For example, the memory bandwidth requirement for VC-1 MP HD 720p 30 fps decoding is up to 378 Mbytes/sec. To support such high memory bandwidth, we employed two memory servers, each of which includes a direct-mapped L2 cache and an SDRAM/DDR2 controller with bank interleaving and prefetching, as shown in Fig. 8. We traded off its performance and area by configuring its parameters including cache size and line-fill length for prefetching. With the memory server, we could hide the access latency of SDRAM or DDR2 and increase effective memory bandwidth with about 22–50%. This bandwidth gain comes from the two features based on cache address mapping: bank-interleaving every SDRAM access as shown in Fig. 9(a) and cache-line with improving spatial locality by mapping more frequently changing address bits into the LSB side as shown in Fig. 9(b), which improves the spatial locality of prefetch access patterns.

Figure 10 shows the bandwidth utilization of the SDRAM memory server. The average memory bandwidth per macroblock required for HD 720p VC-1 decoding is 3.5 kbytes, which corresponds to 378 Mbytes/sec. To satisfy this memory bandwidth requirement with enough room for future extension to the full-HD decoding, we employed two memory servers: one for SDR SDRAM and the other for DDR2 SDRAM. After refining the CATs in the VC-1 video decoder we obtained system architecture with a 32-bit AMBA bus as shown in Fig. 11. Storage for large array channels is assigned to an off-chip memory through a multi-channel configurable memory server. Consequently, about

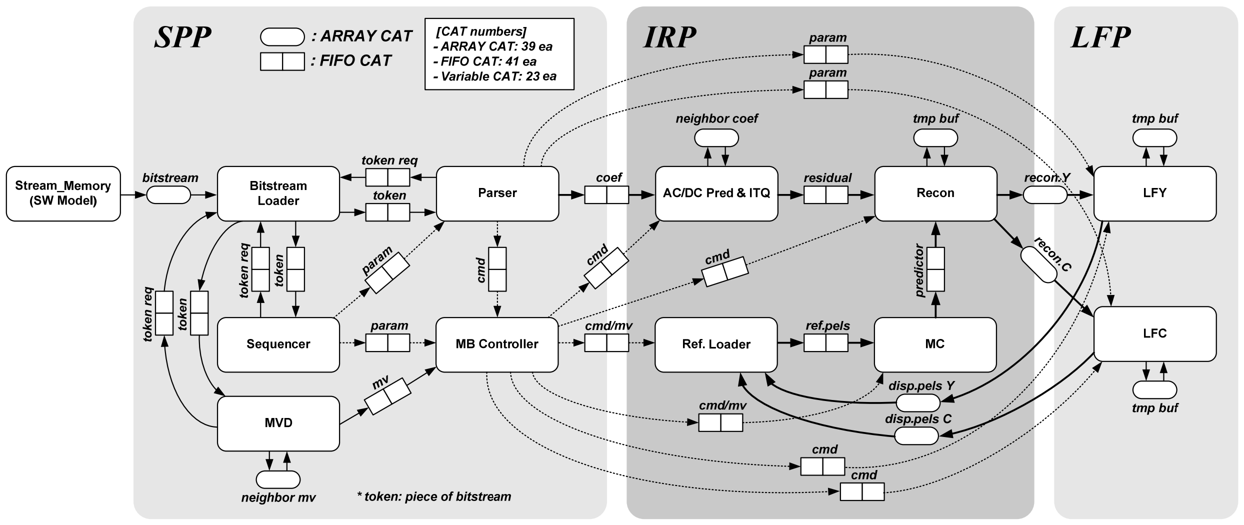


Fig. 6 Simplified TLM model for the VC-1 decoder.

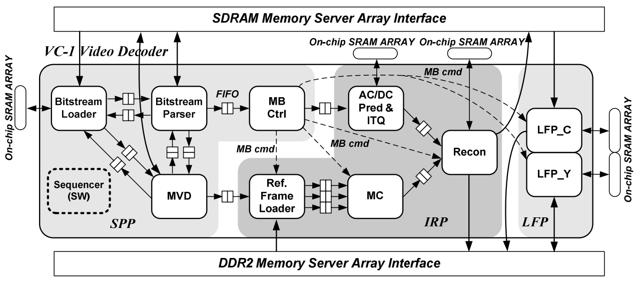


Fig. 7 Simplified block diagram of the VC-1 decoder model after channel refinements.

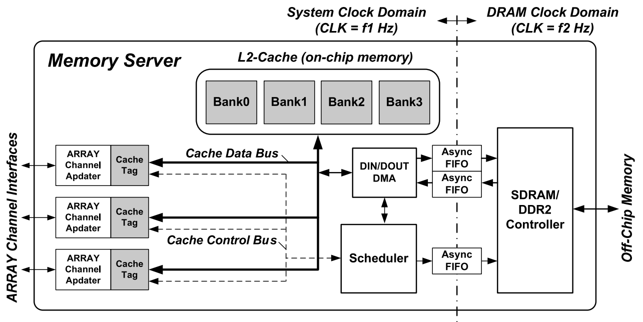


Fig. 8 Architecture of the memory server.

20 large array channels are directly connected to the memory servers in the decoder. Note that the memory servers provide two AHB interfaces for a processor and its peripherals such as a TFT LCD controller and a UART.

4. Computation Block Architectures

In this section, we explain the architecture of computation blocks of the VC-1 decoder in Figs. 6 and 7. Each computation blocks satisfy its throughput requirement such that its average number of cycles per MB should be less than 787 for decoding HD 720p 30 frames per second at the operating frequency of 85 MHz, as shown in Fig. 12.

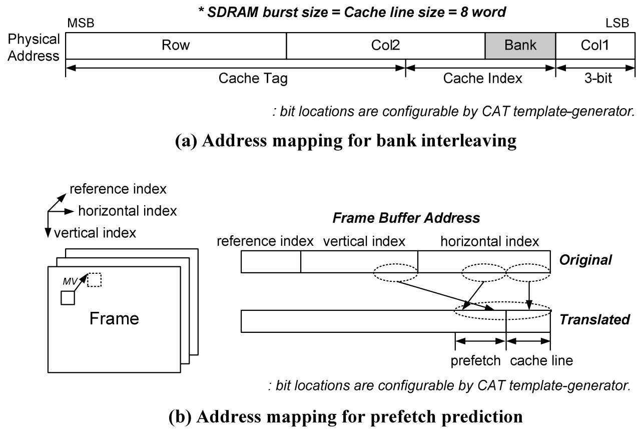

* SDRAM burst size = Cache line size = 8 word
MSB
LSB
Physical Address
Row
Col2
Bank
Col1
Cache Tag
Cache Index
3-bit
: bit locations are configurable by CAT template-generator.
(a) Address mapping for bank interleaving
reference index
horizontal index
vertical index
MV
Frame
Frame Buffer Address
Original
Translated
prefetch
cache line
(b) Address mapping for prefetch prediction

Fig. 9 Address mapping for the memory server.

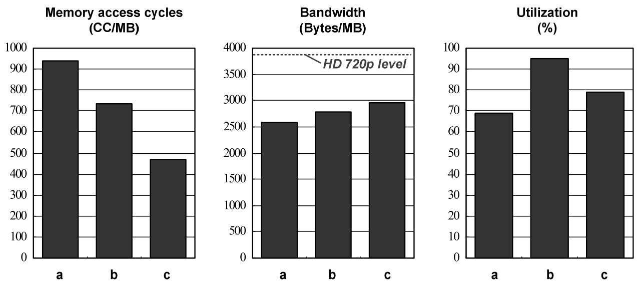


Fig. 10 Average external SDRAM access cycles and transfer-bytes per MB, and bus utilization for the test bitstream MR3_TANDBERG_B of QCIF 10 frames. (a) AHB bus + a SDRAM controller, (b) a memory server CAT with f2=f1, and (c) a memory server CAT with f2=2 × f1. Here, f1 and f2 are the system clock and the SDRAM clock, respectively as shown in Fig. 8.

4.1 Syntax Parsing

The syntax parsing part consists of a software sequencer

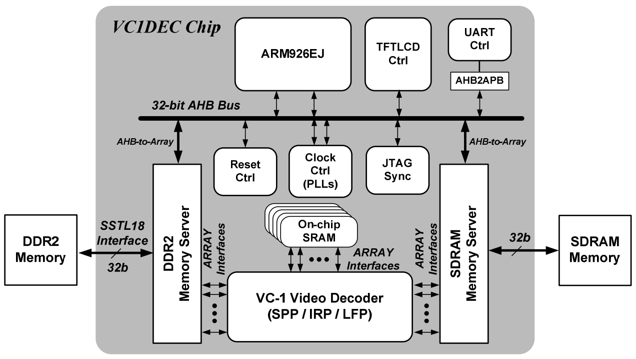


Fig. 11 System-level architecture of the VC-1 decoder.

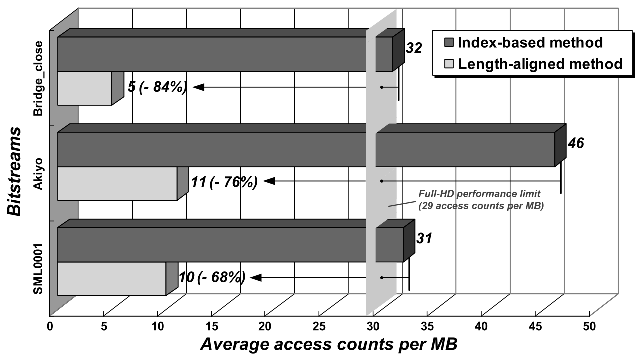


Fig. 13 Average number of VLD table accesses.

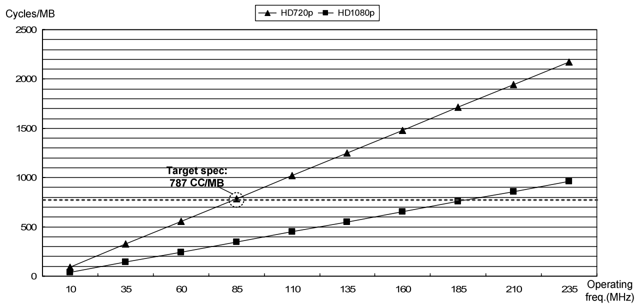


Fig. 12 Operating cycles per MB for HD-level real-time decoding.

and four hardware blocks such as a bitstream loader, a bitstream parser, a motion vector decoder, and a macroblock controller, as shown in Fig. 7. This part's role is to obtain syntax elements and decoding parameters from the VC-1 bitstream.

The bitstream loader gets a token, which has a specified length of bits, according to each request in the three command FIFOs from a sequencer, a bitstream parser, and a MVD. Then the bitstream loader puts tokens into their corresponding token FIFOs. It employs dual buffering and maximizes the burst length of transfers to reduce an access latency of the off-chip memory.

The sequencer obtains picture-level parameters, the motion vector decoder (MVD) calculates motion vectors from motion vector differences and neighbor motion vectors, and the bitstream parser extracts both macroblock-level and block-level syntax elements respectively by parsing a token from their corresponding token FIFOs. By using the motion vectors and picture parameters, the MB controller sends commands to other computation blocks in the other two parts for image reconstruction and loop filtering.

The bitstream parser parses a token by using more than 30 variable length code (VLD) tables. According to the VC-1 standard, each VLD table should be accessed sequentially until a matched code is found. If there is a match, the index of the match VLD code is sent to a run-length decoder (RLD). This sequential code matching cannot provide enough performance at the operating frequency of about 100 MHz for HD-level performance. Moreover, it is not efficient to implement them with random logic to decode every code in one cycle because their circuits are not compact due to their low correlation.

In order to obtain a high-performance parser, therefore we rearranged the VLD tables by assigning the short codes to lower addresses in the ROM. By sequentially comparing a ROM word one by one starting from the lowest address of a VLD table, it is more probable to find the matched code earlier because shorter codes are accessed more frequently. By adopting a ROM word that contains multiple VLD codes, for each access we can compare them in parallel with multiple comparators, which consequently reduce the number of ROM table accesses substantially.

We compared the average number of the table accesses in VLD decoding for two cases: one for the original index-based tables and the other for the re-arranged tables and found that the latter is at least three times faster than the former, as shown in Fig. 13.

Because the rearranged VLD table requires a table index field in addition to the codeword field, its ROM size gets a little larger although its VLD performance can meet the HD-level decoding requirement. To reduce this overhead, we merged four VLD tables for coded-block patterns, motion vector differentials, and AC/DC coefficients into a merged VLD ROM to reduce their control logic.

In the VLD ROM for the re-arranged tables, a ROM word is 232 bits and consists of eight short or four long codes. The short codes are for the VLD codewords of less than or equal to 16 bits while long codes, for the VLD codes of larger than 16 bits which exist only in the VLD table for DC coefficients. Each code has three fields for a VLD codeword, its code size, and its index in the original VLD table. The sizes of fields for a VLD codeword, its size and its index are 16, 5 and 8 bits respectively in each short-type code, as shown in Fig. 14. By employing short and long codes together, we reduced the ROM size by 40.4%.

We also implement the run-level decoder with a merged run-level ROM table by combining three tables for a run-level pair, a delta run, and a delta level into one table as shown in Fig. 15. It has two functions: reading the run-level for an input and generating a sequence of the coefficients. If the run-level result represents the escape mode, the merged run-level table should be accessed again to get a delta level

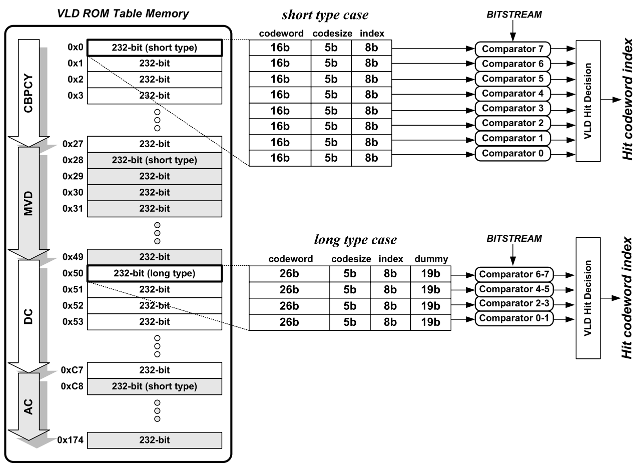


Fig. 14 Merged VLD ROM and its matching circuit.

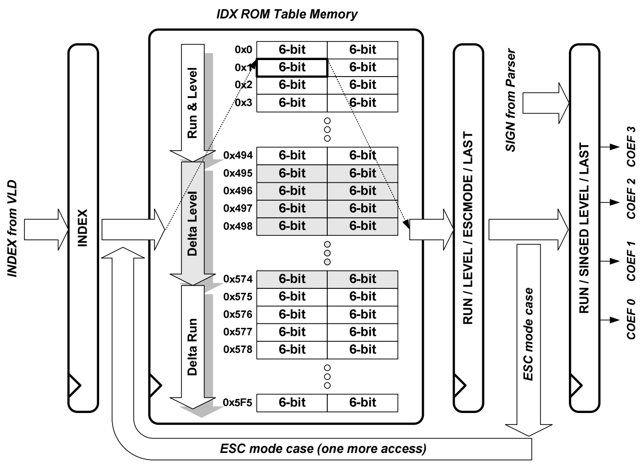


Fig. 15 Data flow of the run-length decoding.

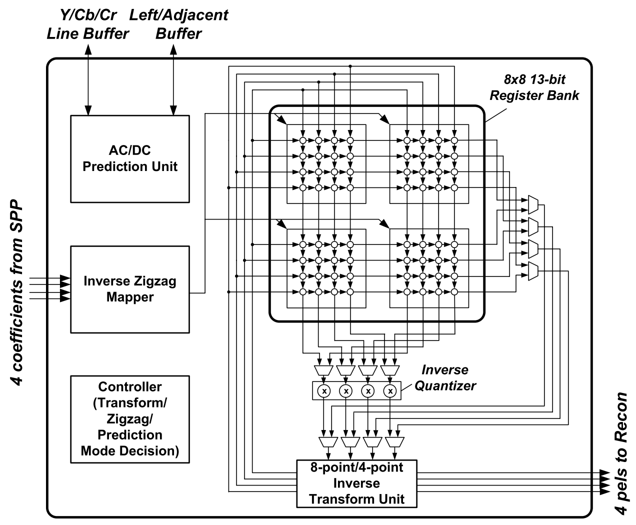


Fig. 16 Block diagram for AC/DC prediction and ITQ.

Table 1 Sequence of AC/DC prediction, IZ and ITQ for four modes.

Mode	Operation	Pipeline Cycles	Range
INTRA 8x8	DC Pred.	6~ cycles	Min. 50 cycles
	AC Pred.	6~ cycles	
	IZ	1~16 cycles	
	AC/DC Recon.	2 cycles	
	Line Buf. Update	4~ cycles	
	Row IT	18 cycles	
	Adj. Buf. Update	4~ cycles	
	Column IT	18 cycles	
INTER 8x8	IZ	0~16 cycles	36~52 cycles
	Row IT	18 cycles	
	Column IT	18 cycles	
INTER 8x4, 4x8	IZ	0~8 cycles	20~28 cycles
	Row IT	10 cycles	
	Column IT	10 cycles	
INTER 4x4	IZ	0~4 cycles	11~15 cycles
	Row IT	6 cycles	
	Column IT	5 cycles	

* IZ : Inverse Zigzag, IT : Inverse Transform, Adj : Adjacent

or a delta run. The RLD generates four coefficients simultaneously, which are sent to the image reconstruction part. The average performance of the syntax parsing part is 402 cycles per MB, which can decode a sequence of HD 720p 30 fps images at the operating frequency of 44 MHz.

4.2 Image Reconstruction

The image reconstruction part of the VC-1 decoder consists of an ITQ block, a reference frame loader, a motion compensator, and an image reconstructor, as shown in Fig. 7. Each macroblock of an image is sequentially reconstructed by the reconstructor, which adds its residuals obtained from the inverse transform/quantization (ITQ) block that also performs AC/DC prediction and inverse zigzag operation to the predicted values of the macroblock. The motion compensator takes the results of the reference frame loader and the motion vectors as its inputs and generates the predicted values for the macroblocks. The reference frame loader load a macroblock from the reference frame by using the integer parts of the motion vectors. We will explain these three blocks one by one in the following.

To share four 4×4 sub-block registers, we merged all the operations for AC/DC prediction, inverse zigzag, inverse quantization and inverse transform into the ITQ block, as shown in Fig. 16.

The ITQ block has four operation modes: one for an intra-block prediction for the $8{\times}8$ blocks, and three for inter-block prediction of different block sizes: 8×8, 4×4, and $4 \times 8/8 \times 4$. Its internal modules are fully pipelined so that the ITQ block gets four coefficients per cycle as its input and generates four residuals every cycle. Its summarized operation sequences for the four modes are shown in Table 1. In the INTRA $8{\times}8$ mode, AC/DC prediction is first performed by using coefficients from the three upper-line buffers in SDRAM and coefficients from the left-neighbor buffer in an on-chip SRAM buffer. An inverse zigzag block maps each coefficient into $4{\times}4$ register banks according to one of seven inverse zigzag patterns that depends on the block type, transform type and prediction direction. After inverse zigzag mapping, AC/DC coefficients are reconstructed, which is followed by updating the line buffers. Then, row ITQ operations are performed sequentially, which is followed by updating adjacent buffer. Finally, column ITQ operations are performed sequentially. To hide the latency of off-chip

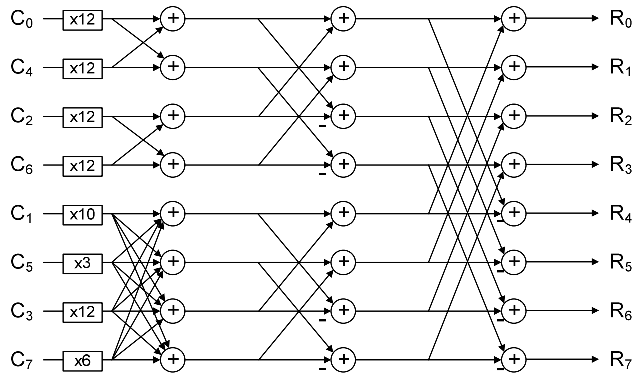


Fig. 17 Butterfly operations for inverse transform.

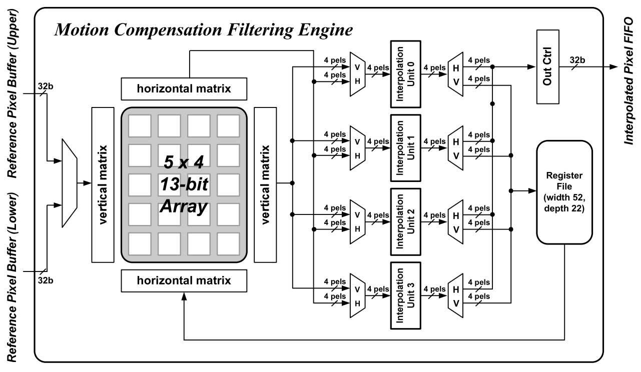
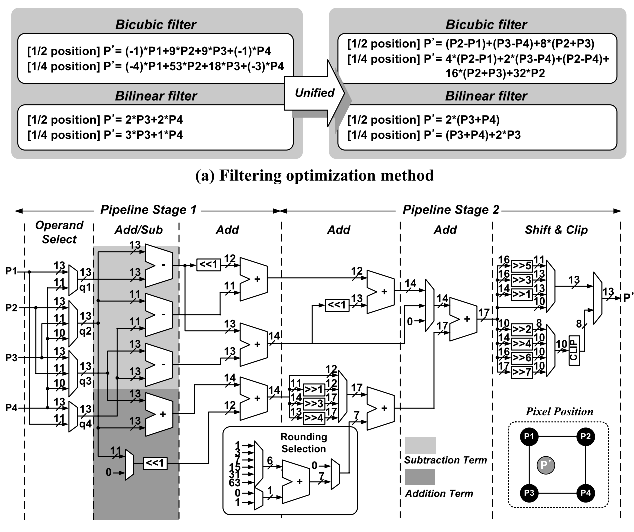


Fig. 18 Datapath of the motion compensator.


Bicubic filter
[1/2 position] P' = (-1)*P1+9*P2+9*P3+(-1)*P4
[1/4 position] P' = (-4)*P1+53*P2+18*P3+(-3)*P4
Bilinear filter
[1/2 position] P' = 2*P3+2*P4
[1/4 position] P' = 3*P3+1*P4
Unified
Bicubic filter
[1/2 position] P' = (P2-P1)+(P3-P4)+8*(P2+P3)
[1/4 position] P' = 4*(P2-P1)+2*(P3-P4)+(P2-P4)+16*(P2+P3)+32*P2
Bilinear filter
[1/2 position] P' = 2*(P3+P4)
[1/4 position] P' = (P3+P4)+2*P3
Pipeline Stage 1
Pipeline Stage 2
Operand Select
Add/Sub
Add
Shift & Clip
P1 P2 P3 P4
Rounding Selection
Subtraction Term
Addition Term
Pixel Position
P'

(a) Filtering optimization method

(b) Datapath of the interpolation unit

Fig. 19 Datapath of an interpolation unit for motion compensation.

memory access, buffer updating and ITQ operations are parallelized, as shown in Table 1.

The inverse transform of VC-1 is defined by the following matrix equations [1]:

$$E_{M\times N} = (D_{M\times N} \cdot T_M + 4) \gg 3$$
$$R_{M\times N} = (T_N^T \cdot E_{M\times N} + C_N \cdot 1_M + 64) \gg 7$$

where $E_{M\times N}$ is the intermediate matrix obtained by the 1D inverse transform and $R_{M\times N}$ is the resultant matrix of the 2D inverse transform. $D_{M\times N}$ is an input matrix of the inverse transform, which is computed by the IQ. C_N is a column vector of dimension N and 1_M is a row vector of dimension M. The matrices T_M and T_N^T contain the inverse transform coefficients.

In our computation block for the inverse transform, we employ architecture with butterfly operations to reduce the number of multiplications and additions as shown in Fig. 17. By using the modified butterfly scheme, we reduced the complexity of ITQ logic substantially, which requires only 192 multiplication and 384 additions for an 8×8 block inverse transform.

Through optimized pipelining, the average performance of the ITQ block is 300 cycles per MB, which can decode a sequence of HD 720p 30 fps images at the operating frequency of 33 MHz. It outperforms an ITQ design [9] by 78%.

Inter-prediction is performed by a reference frame loader and a motion compensator, as shown in Fig. 7. The reference loader gets a block of reference pixels from the off-chip memory using an integer-pel motion vector and writes them to a local SRAM buffer. The motion compensator calculates the predicted pixel values using a fractional-pel motion vector and the reference pixels. The reference frame loader contains a transposing unit for vertical interpolation and an intensity compensation unit. The transposing unit has a four-bank register file for address interleaving to hide the external memory latency. Moreover, by separating the reference pixel buffer into the upper and lower banks, loading reference pixels and reading them for motion compensator (MC) can be executed in parallel.

The MC consists of four interpolation units, a 5×4 array of 13-bit registers, and a 22×4 pel 13-bit register file as shown in Fig. 18. The MC reads input data into the 5×4 register array from one of two reference pixel buffers in the frame loader. The register array can shift the pixels either horizontally or vertically. Each of four interpolation units can read either horizontal or vertical of pixel data from the 5×4 register array and perform either bicubic or bilinear interpolation. The 52×22 buffer stores intermediate results of vertical or horizontal interpolation. Note that in VC-1 motion compensation vertical interpolation should be performed before horizontal interpolation. Vertical and horizontal filtering can be skipped for some macroblocks, which depends on their fractional motion vectors and their positions in the frame.

In order to optimize the interpolation units, we designed a unified interpolation filter that can perform for various modes such as bicubic/bilinear filtering, vertical/horizontal filtering, and one-way/two-way filtering. Its datapath, which efficiently performs both bicubic and bilinear filtering, is simplified by exploiting addition and subtraction terms, as shown in Fig. 19 and requires only 11 adders and several shifters. The MC block achieves the average

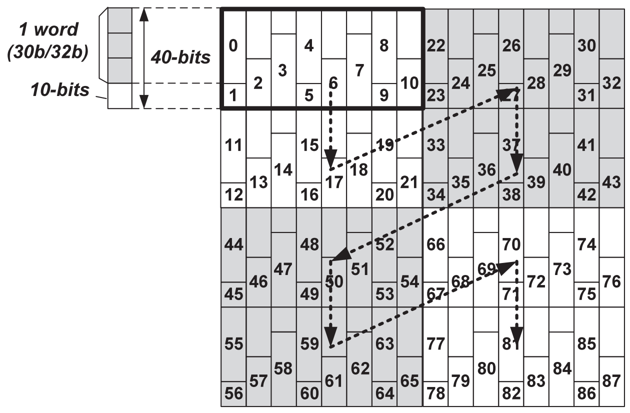


Fig. 20 Bamboo pattern for storing the packed words of the 10-bit reconstructed pixels for a 16 × 16 luminance macroblock.

performance that can decode a sequence of HD 720p 30 fps images at the operating frequency of 37 MHz.

Just like the MC and ITQ blocks, architecture of the image reconstruction block is also tuned to compute four vertical reconstructed pixels together. Therefore, it reads four 8-bit predicted pixels and their corresponding 9-bit residuals in the FIFO channels from the MC and ITQ blocks, respectively, and simply add them to reconstruct four 10-bit unclamped pixels, which are transferred to the external SDRAM. To minimize fragmentation of the 10-bit pixel data in byte addressing of the SDRAM, we decided to bundle three 10-bit pels into a 32-bit word. Therefore, together with the packed word, its output pattern for a macroblock is written with a bamboo pattern shown in Fig. 20. Its write access sequence is zigzagged to simplify the logic for the read access sequence of the loop filter engine.

4.3 Loop Filtering

The loop filtering engines perform both overlap smoothing and deblock filtering to soften the blocking artifacts due to block-level transform and prediction. According to the VC-1 standard, all overlap smoothing for vertical edges should be performed before all overlap smoothing for horizontal edges in the frame level, and then all deblock filtering for horizontal edges are performed before all deblock filtering for vertical edges in the frame level.

Because of these filtering order constraints, the MB-level algorithms for H.264 loop filtering [10], [11] cannot be applied to VC-1 loop filtering. To reduce the required filtering buffer size, therefore we transformed the frame-level filtering algorithm to a MB-level one that satisfies the frame-level filtering order constraints. Figure 21 shows the architecture for loop filter engines we employed, which can efficiently perform 4-pel edge filtering according to the filtering orders illustrated in Fig. 22. Note that the VC-1 decoder employs two filtering engines: one for luminance data and the other for chrominance data.

The luminance loop filtering engine performs filtering for all the edges numbered within an extended macroblock

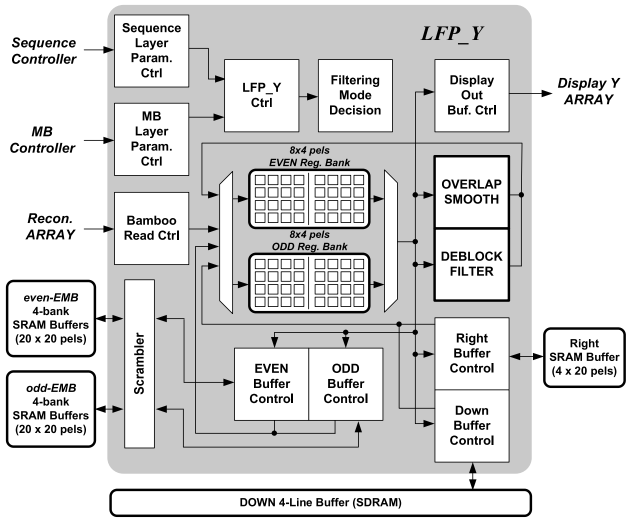


Fig. 21 Architecture of the luminance loop filtering engine.

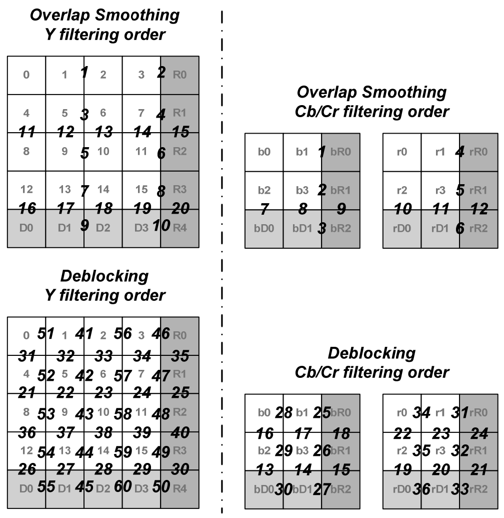

Overlap Smoothing Y filtering order
Overlap Smoothing Cb/Cr filtering order
Deblocking Y filtering order
Deblocking Cb/Cr filtering order

Fig. 22 Filtering order for edges of the boundaries of 4 × 4 sub-blocks in a buffer that stores a macroblock together with right and down sub-blocks. (For a left-top case)

buffer (EMB) including a macroblcok and its right and down sub-blocks, as shown in Fig. 21. The engine requires two (even and odd) EMBs with 4 single-port banks that store both the input pixels and intermediate filtering results. Note that a loop filtering engine get a 3-pel packed word from the image reconstructor and unpack and move it into one of the two (even and odd) 8×4 internal registers that can transpose data. The size of each EMB is an array of 20 × 20 10-bit pels, which can store 25 4 × 4 blocks. The filtering engine employed two EMBs for ping-pong operation to hide the access latency of reading the reconstructed pixels stored in the external SDRAM memory, which is as shown in Fig. 24. There are 9 cases for filtering edge patterns, each of which skips different edges according to the current MB position. Just for the upper-left corner case, the orders for two filtering

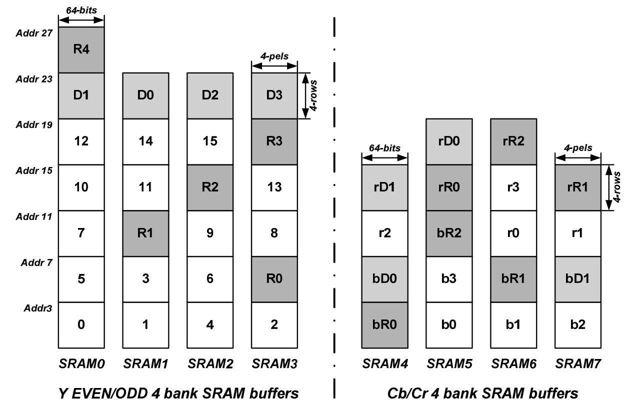


Fig. 23 Scrambling scheme of 25 sub-blocks into 4 banks of a buffer.

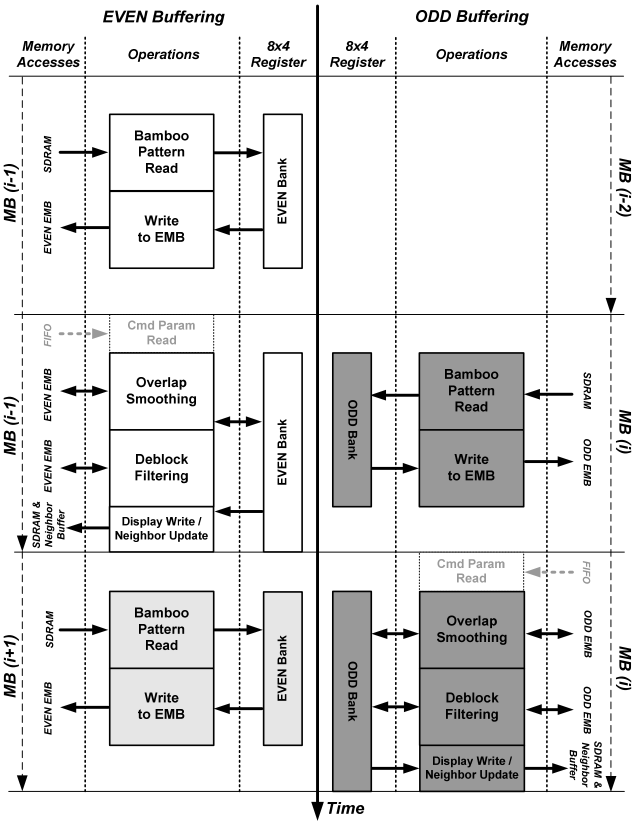


Fig. 24 Ping-pong buffering operation of the luminance LFP in a time-domain diagram.

sequences for luminance and chrominance that satisfy the VC-1 specification are shown in Fig. 22.

Moreover, pixel data for a pair of sub-blocks should be moved into one of the 8×4 buffers for filtering without a stall for high performance. To eliminate pipelining stall of a group of four row- or column-filtering operations, we scrambled all the 25 sub-blocks into four banks of a single-port SRAM buffer as shown in Fig. 23. For example, vertical edge 3 between blocks 5 and 6 is filtered for overlap smoothing and vertical edge 42 for deblocking. Therefore, blocks 5 and 6 should be stored in two different banks of the buffer so that the filter engine can fetch a pair of 4 pels every cycle from the local buffer.

Moreover, we simplified the loop filtering engine by sharing the control circuits for both overlap smoothing and deblock filtering as shown in Fig. 21. And, two (even and odd) 8×4 internal registers are used for temporary storage of unpacked reconstructed pixels, all the inputs (outputs) of filtering from (to) an EMB as shown in Fig. 24. Note that except buffering the parameters of the input commands, the inputs and outputs of all internal operations are buffered in a ping-pong style buffer with two 8×4 registers. The LFP takes from 152 to 560 cycles for a macroblock, which average is 424 cycles.

Furthermore, we reduced the required cycles of the loop filtering operations to 252 cycles by employing two loop filtering engines: one for luminance and the other for chrominance. We obtained the average performance of decoding a sequence of HD 720p 30 fps images at the operating frequency of 46 MHz.

5. Verification and Implementation

For efficient verification of the computation blocks, we employed the Syste mVerilog environment [12] together with three advanced techniques such as assertion-based verification (ABV) [13], constrained-random verification (CRV) [14], and coverage-driven verification (CDV) [14]. Furthermore, to fill the coverage holes of natural image sequences and video parameters, we used a constrained-randomly generated image sequence as a test sequence. With these verification methods, we achieved a higher confidence level of the design. For example, statement, branch, condition, and toggle code-coverages for QCIF 300 frames of Akiyo, an image sequence of the conformance test are 58.2%, 53.3%, 27.3%, and 58%, respectively; those for the constrained-randomly generated one are 81.6%, 77.9%, 56.0%, and 88.5%, respectively, which are measured by Questa SV/AFV [15]. Moreover, we achieved full function coverage with CRV and CVD which cannot be done with conformance tests.

The target specification of the VC-1 decoder is to decode HD 720p sequence with 30 fps at the operating frequency of 85 MHz with minimal silicon area. To achieve the target performance and area, we tried various configurations for communication DSE. To summarize it with a simpler table, however we just picked only eight meaningful ones especially for 14 array CATs that strongly affect the performance and area. The DSE result is summarized in Table 2, where each column represents a configuration with 14 numbers. Each number means a uniquely refined array CAT; (1) one with an on-chip SRAM, (2) one with a SDRAM controller, (3) one with a DDR2 controller, (4) one including a DDR2 controller with prefetching 8-words, (5) one including a DDR2 controller with prefetching 16-words, and (6) one including a DDR2 controller with prefetching 24 words.

The performance and area for each configuration is illustrated in Fig. 25, where only four configurations such as

Table 2 Eight configurations for array CATs.

Array Name	C1	C2	C3	C4	C5	C6	C7	C8
Stream	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)
MVtype	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)
Coeff	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
ITQ	(2)	(1)	(2)	(2)	(2)	(2)	(2)	(3)
MV	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)
Ref.load	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
Recon	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
Blk info	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)
Nz info	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)
Recon Y	(4)	(3)	(3)	(3)	(6)	(5)	(2)	(3)
Recon C	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)
LF local	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
LF line	(2)	(2)	(1)	(2)	(2)	(2)	(2)	(3)
Disp	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(3)

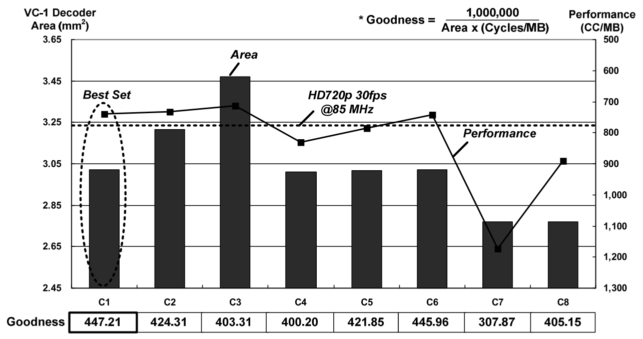


Fig. 25 Simulation results for various memory configurations.

Table 3 Complexity and performance of the computation blocks in the VC-1 decoder.

Components		Gate Counts	On-chip SRAM	Performance (Cycles/MB)
SPP	Bit-stream Loader	4.8k	128 B	402
	Bit-stream Parser	10.4k	128 B[1)]	
	MVD	7.8k	0 B	
	MB Ctrl	4.9k	64 B	
IRP	AC/DC Pred.&ITQ	32.8k	72 B	339
	Ref. Loader	11.0k	0 B	
	MC	11.3k	143 B	
	Recon.	3.9k	232 B	
LFP	LFP Y	66.5k	1.7 KB	424
	LFP C	41.7k	696 B	
Total complexity of the computation blocks		195.1k	3.1 KB	518

1) The ROM of 13.1 KB is not included

C1, C2, C3, and C6 meet the target performance. We implemented C1 into silicon because its area is the smallest among the four configurations.

Tables 3 and 4 shows the synthesis result and performance for each computation block of the configuration C1. In Table 3, the gate counts are obtained from the timing constraint of 133 MHz for a 0.13 μm CMOS process, and the effect of the external memory latency is not included yet in the computational performances.

According to the simulation result for a conformance bitstream SML0001, which includes external memory latency, the average performance of the VC-1 decoder is average 740 cycles per MB, which can decode HD 720p bitstream with 30 fps at 80 MHz. Note that its performance margin is 47 cycle per MB, compared to the target performance.

Table 4 Complexity of the communication channels in the VC-1 decoder.

Components	Gate Counts	On-chip Memory
Quantized Coefficient FIFO	-	192 B
Residual FIFO	-	240 B
Register FIFOs	15.2k	-
Ref. Pel Buffers	-	264 B
AHB Bus	3.2k	-
SDR memory server	72.5k	2 KB
DDR2 memory server	78.8k	8 KB
etc	51.6k	112 B
Communication Total	221.3k	10.8 KB

Table 5 Complexity of computation and communication parts in the VC-decoder.

Part	Logic Area	On-chip Memory Area
Computation	195.1k	16.2 KB
Communication	221.3k	10.8 KB
Total	416.4k (3.03 mm^2)	27.0 KB (2.06 mm^2)

Table 6 Comparison of VC-1 decoders.

Item	Proposed	Design[16]	Design[17]	Design[18]
Standard	VC-1 Dec	VC-1 Dec	VC-1 Dec	VC-1 Dec
Target	HD720	CIF	D1	D1
Profile	Main	Simple	Advanced	Main
Frame rate	30 Hz	15 Hz	30 Hz	30 Hz
Gate count	262K	-	-	-
Memory	27 KB	-	-	> 20.1 KB
Core Area	5.1 ㎟	-	-	-
Op. Freq.	80 MHz	-	310 MHz	54 MHz
Cycles/MB	740	4,629	7,652	1,280
Process	130 nm	90 nm	90 nm	-
Power	230 mW	-	-	590 mW

In designing the VC-1 decoder we manually coded only the RTL models for the computation components, which correspond to 47% of the total logic size, as shown in Table 5. Note that all the communication components are obtained simply by configuring the CATs selected from the channel template library in SoCBase-DE. Although the total logic complexity of the VC-1 decoder is 416,400 gates in Table 5, it becomes about 261,900 gates if the two memory server and the AHB bus are excluded in its communication part.

Table 6 compares the proposed decoder with several other VC-1 decoders. Because VC-1 is a relatively new standard, there are only a few published works. The VC-1 decoders in [16] and [17] are a software solution using OMPA2420 and DM6437, respectively. Just for VC-1 decoding, the performance of the proposed design is 42% better than that of the design in [18]. Note that the latter is a VLSI implementation for a multi-format video codec, which supports only decoding for VC-1 and both encoding and decoding for H.264 and MPEG-4.

For verification purpose, we prototyped the VC-1 de-

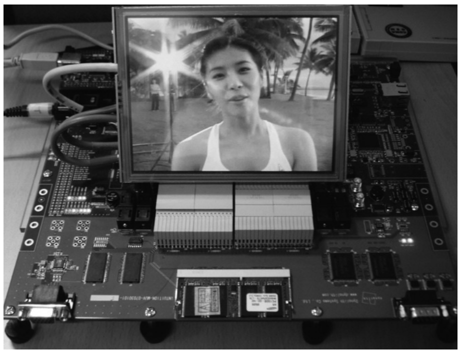


Fig. 26 FPGA prototyping board.

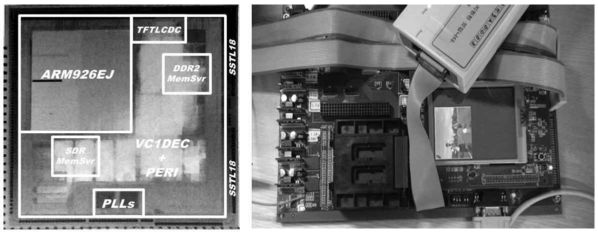


Fig. 27 Micrograph of the VC-1 decoder and its working die test.

coder on a FPGA board with a Xilinx Vertex-4 chip, as shown in Fig. 26. The prototype can decode about 30 fps in real time for VGA sequence at 33 MHz. As shown in Fig. 27, we have implemented a VC-1 decoder chip for HD 720p 30 fps with 0.13 μm CMOS technology. The VC-1 decoder chip is pad-limited and its area is 25 mm^2 including the pads. The supply voltage is 1.2 V for its core and 3.3 V for IOs and its maximum operating frequency is 133 MHz in the typical condition.

6. Conclusions

In this paper, we described a high-performance VC-1 main profile decoder for HD video applications. With the system-level and component-level optimizations from algorithmic and architectural perspectives, the proposed VC-1 design can achieve real-time decoding on HD 720p video (1280 $\times$ 720 @30 Hz) when operating at 80 MHz. The proposed design occupies only 5.1 mm^2 core area, which requires 261,900 gates with 13.9 KB SRAM and 13.1 KB ROM for a 1P8M 0.13 μm CMOS process.

Acknowledgments

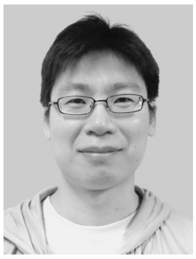
This work was supported by Center for SoC Design Technology (CoSoC), Inter-university Semiconductor Research Center (ISRC) of Seoul National University, and BK21, SystemIC 2010, IP/SoC Center of Seoul, Korea. Chip fabrication was supported by SMIC via KETI/IPCoS.

References

[1] SMPTE 421M-2006, "VC-1 compressed video bitstream format and decoding process," Feb. 2006.
[2] ISO/IEC 14496-10, "H.264/MPEG-4 Part 10," May 2003.
[3] S. Park, S. Yoon, and S.-I. Chae, "A mixed-level simulation environment for refinement-based design methodology," Proc. RSP, June 2006.
[4] S. Park, S.Y. Yoon, and S.-I. Chae, "Reusable component IP design using refinement-based design environment," Proc. ASP-DAC, pp.588–593, Jan. 2006.
[5] IEEE 1666-2005, "IEEE standard systemc language reference manual," March 2006.
[6] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.23, no.6, pp.952–961, June 2004.
[7] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, Surviving the SoC revolution; A guide to platform-based design, Kluwer Academic Publisher, 1999.
[8] "Windows media video 9 algorithm overview," MicroSoft Windows Media 9 Series, Windows_Media_Video_9_1.ppt
[9] S. Lee and K. Cho, "Circuit implementation for transform and quantization operations of H.264/MPEG-4/VC-1 video decoder," Design & Technology of Integrated Systems in Nanoscale, DTIS, Sept. 2007.
[10] C.C. Cheng, T.S. Chang, and K.B. Lee, "An in-place architecture for the deblocking filter in H.264/AVC," IEEE Trans. Circuits Syst. II, Express Briefs, vol.53, no.7, pp.530–534, July 2006.
[11] T.C. Chen, S.Y. Chien, Y.W. Huang, C.H. Tsai, C.Y. Chen, and L.G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," IEEE Trans. Circuits Syst. Video Technol., vol.16, no.6, pp.673–688, June 2006.
[12] Accellera, "SystemVerilog 3.1a," Language Reference Manual, 2004.
[13] D.L. Perry and H.D. Foster, Applied Formal Verification, McGraw-Hill, New York, 2005.
[14] P. Dasgupta, A Roadmap for Formal Property Verification, Springer, 2006.
[15] MentorGraphics, "QuestaTM SV/AFV user's manual," Aug. 2006.
[16] R. Lakshmish, K.Y. Praveen, K. Maiti, J. Pai, and T.K. Adhikary, "Efficient implementation of VC-1 decoder on Texas Instrument's OMAP 2420-IVA," Systems, Signals and Image Processing, June 2007.
[17] Texas Instruments, "Windows media VC-1 advanced profile decoder (v1.02) on C64x+," SPRS 437, June 2007.
[18] M. Hase, K. Akie, M. Nobori, and K. Matsumoto, "Development of low-power and real-time VC-1/H.264/MPEG-4 video processing hardware," Proc. 12th Asia and South Pacific Conference on Design Automation, pp.637–643, Jan. 2007.

Jinhyun Cho received the B.S. degree in electronics engineering from Kyoungpook University, in 1997, and the M.S. degree from Sungkyunkwan University, in 2002. During 1997–2005, he engaged in SystemLSI Division of Samsung Electronics Company, Ltd. He is currently working toward the Ph.D. degree in electronic engineering from Seoul National University. His major research interest includes MPEG-4, H.264/AVC, and VC-1 video processing and advanced verification methodology for complex SoC design.

Doowon Lee received the B.S. and M.S. degrees in electronics engineering from Seoul National University in 2006 and 2008. He is currently working as professor at the Naval Academy of Korea. His major research interest includes VC-1, H.264/AVC, MPEG-2 video processing.



Sangyong Yoon received the B.S. and M.S. degrees in electronics engineering from Seoul National University in 2000 and 2002, respectively. He is currently working toward the Ph.D. degree in electronic engineering from Seoul National University. His major research interest includes an advanced memory controller design.

Sanggyu Park received the B.S. degree in electronics engineering from Yonsei University in 2000 and received the Ph.D. degree in electronics engineering from Seoul National University in 2008. He is currently working at Samsung Electronics Company, Ltd. His major research interest includes configurable processor and advanced design environment.

Soo-Ik Chae is Professor in Department of Electrical Engineering and Computer Science and Director of Center for SoC Design Technology, Seoul National University, Seoul, Korea. He received the B.S. and M.S. degrees in Electrical Engineering from Seoul National University in 1976 and 1978, respectively, and the Ph.D. degree from Stanford University, Stanford, California, U.S.A. in 1987. He was Instructor of electronics engineering at Korean Air Force Academy from 1978 to 1982. He joined Seoul National University in 1990. He had been Director of Inter-university Semiconductor Research Center, Seoul National University from Mar. 2001 to Feb. 2003. His primary research interests are focused on system-level designs including transaction-level design for video and graphics systems.