An Efficient Implementation of Multi-channel H.264 Decoder SoC

Wonjong Kim[1], Juneyoung Chang[1], and Hanjin Cho[1]
[1] SoC R&D Group, ETRI
Gajeong-Dong 161, Yusong-Gu, Daejeon 305-700, Korea
E-mail: wjkim@etri.re.kr , jychang@etri.re.kr, hjcho@etri.re.kr

Abstract: We developed a multi-channel H.264 decoder based on a single channel H.264 decoder which was developed using autonomous module design methodology. Since an autonomous module can control itself by checking states of its neighbor modules, we could easily extend the decoder for decoding multi-channel streams. We utilized the nature of SDRAM structure for efficient use of frame data. We developed specialized SDRAM controller and DMA controller for efficient data transfers of 2-dimensional data. By assigning dedicated channels for modules which require data transfer from/to SDRAM, they can freely use SDRAM data. The decoder can decode 1~16 channels of QVGA, 1~4 channels of VGA, 1~2 channels of XGA or HD, or 1 channel of Full-HD videos at 30 fps within 150 MHz.

1. Introduction

Recent surveillance systems use IP cameras which can encode video data locally and send compressed video data to the centralized control system through Ethernet network [1]. The centralized control system manages data gathering, recording and monitoring by decoding several encoded streams coming from several IP cameras. As the demand for high-definition resolution is increased H.264 is getting more interest for surveillance systems [2-3]. With MPEG4, PC-based control systems can handle about 40 channels of CIF-size encoded streams. However it is difficult to decode more than 20 channels of H.264 encoded CIF-size streams in current PC-based systems. Therefore a hardware-accelerated decoding is necessary for H.264-based surveillance systems.

H.264 is now an emerging standard of video compression and is used in several multimedia applications such as DMB systems, Blue-ray DVDs, and surveillance systems. It was developed jointly by ITU-T and ISO/IEC [4]. The data coding efficiency of H.264 is several times better than that of previous standards. It provides improved coding efficiency and flexibility for network applications. It has some additional features like variable block size, intra prediction, in-loop filter and 4x4 block transform. Those tools make the coding algorithm more complex and make developers spend more cost like design complexity, large amount of calculation, memory bandwidth and power consumption. H.264 has many calculations and lots of encoding mode, so the design complexity is not avoidable. But, it is possible to optimize the algorithm using statistic feature.

In this paper we present an extended design of multi-channel H.264 decoder from a single-channel H.264 decoder. By utilizing autonomous module design methodology proposed in [5], we could efficiently extend the single-channel H.264 decoder focusing on only local modifications for extension. Since H.264 decoder requires a lot of data transfers with frame memory, we proposed an efficient architecture for data transfers with SDRAM. Also we developed specialized DMA controller and SDRAM controller for efficient transfer of 2-dimensional data.

The rest of this paper is organized as follows. We review the single-channel H.264 decoder architecture in section 2. In section 3, we describe the extension of it to multi-channel H.264 decoder. In section 4, we explain some experimental results. Finally concluding remarks are described in section 5.

2. Single-channel H.264 Decoder

Most signal processing hardware uses pipelined architecture for performance and efficiency. For efficient pipelining, each pipeline stage should have similar operation cycle. However, some applications make it very difficult in nature. For example, H.264 decoder has several operation modes: 13 prediction modes for intra frames and 7 types for inter-frame (P-frame) motion estimation. Therefore, each module has various operation cycles for each macroblock (MB) mode. To overcome this problem, each module should operate adaptively for various MB modes which can provide adaptive pipelining.

Figure 1 shows the single-channel H.264 decoder architecture using autonomous module design methodology described in [5]. Dotted lines are DMAC channels dedicated to each module by which modules can request data transfer from/to SDRAM.

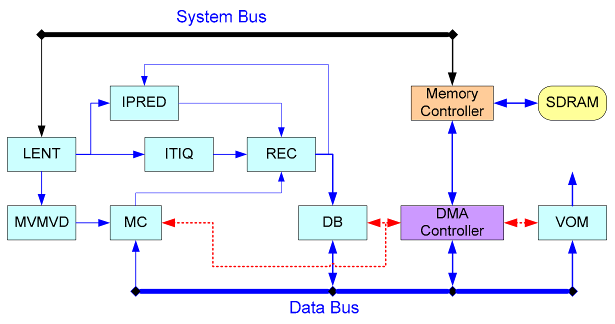


Figure 1. H.264 decoder architecture with autonomous modules.

This work was partly supported by the IT R&D program of MIC/IITA [2008-S-001-01, A Smart Eye based on Convergence SoC]

An autonomous module can control itself by checking states of neighbor modules. It communicates with preceding modules and following modules by using handshaking protocol. It starts operation when all the input data are ready from preceding modules and output data when following modules are ready to receive data.

A simple module consists of 3 parts: data input, data processing, data output, and operation control. Data input and output parts handles data input from the previous module(s) and data output to the next module(s), respectively. Data processing part processes input data to generate output data with given algorithm. Operation control part monitors data input/output parts and controls data processing parts. Some modules may have another part for parameter input which handles parameters for various operation modes of the module. Parameters can be input from the system processor or previous module(s). Figure 2 shows some example module structures. A simple module with only one operation mode has only data input. However a complex module with various operation modes has data input as well as parameter input. Operation control part monitors also parameter input and sets data processing part for necessary operation mode.

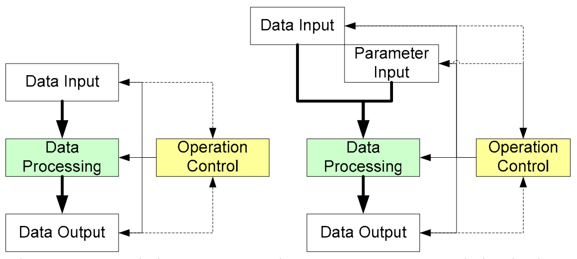


Figure 2. Module structures in autonomous module design.

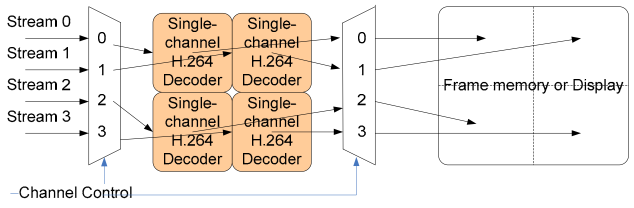
In this design methodology, the system processor may initially set up registers of each module for proper operation and then cares only for primary input/output data. Since designer can concentrate on the communication between related modules, they can easily improve the system performance by modifying critical modules.

Autonomous modules provide distributed control for both data-path and control-path so that they can be very efficient for complex multimedia systems with highly parallel operation. Also they can provide globally asynchronous and locally synchronous (GALS) designs where each module can operate with different clock frequencies [6].

Since it was developed to decode only one stream of encoded video data, it should be modified or extended to decode multiple streams. On behalf of autonomous module design, we could focus on local modifications for related parts for multi-channel decoding activity.

3. Multi-channel H.264 Decoder

Multi-channel decoding can be achieved by using time multiplexing with one decoder or using multiple copies of decoders. Using multiple decoders is an easy solution to be implemented but it increases area cost as shown in Figure 3. Time multiplexing with one decoder does not increase area cost but it requires higher frequency. However in terms of energy consumption both solutions may consume similar amount of energy for decoding the same data. Therefore time multiplexing will be more preferable solution. Both approaches can be mixed if time multiplexing is limited by the maximum achievable clock frequency. In this paper we focused on time multiplexed multi-channel decoding architecture.


Stream 0
Stream 1
Stream 2
Stream 3
0
1
2
3
Single-channel H.264 Decoder
Single-channel H.264 Decoder
Single-channel H.264 Decoder
Single-channel H.264 Decoder
0
1
2
3
Frame memory or Display
Channel Control
Figure 3. Multi-channel decoder with multiple decoders.

3.1 Multi-channel H.264 decoder architecture

Since H.264 decoder can use multiple reference frames, the decoder has a set of registers pointing current and reference frames. We can support multi-channel stream by adding sets of registers for additional channels. However assigning channel number to the highest address bits, we can achieve it by only increasing small number of bits for the channel number.

For the multi-channel decoding, input streams are multiplexed frame by frame. Channel number is embedded in the frame header. Modules with no relation to frame memory process the stream as the same manner as single channel decoding. Modules handling frame memory refer to the set of registers with channel number. Actually there is no major change in the architecture but only small modifications to modules related to frame memory such as LENT, MC, DB and VOM in Figure 1. LENT is entropy decoding module and is modified to decode frame header and to find the channel number. MC (motion compensation module) and DB (deblocking filter module) use the channel number to find the frame memory location for the channel. Since VOM (video output module) outputs decoded frame data after one set of frames is fully decoded, it uses appropriately delayed channel number. The decoder frequency should be increased as the number of channels increases.

The decoder outputs each frame to designated region of the frame memory or display for the system as shown in Figure 4 for 4-channel streams.

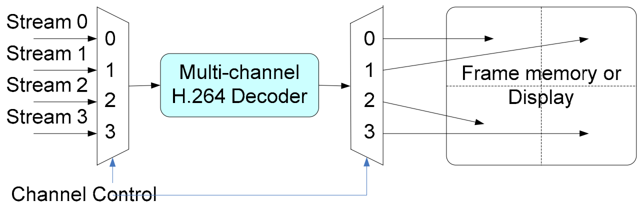
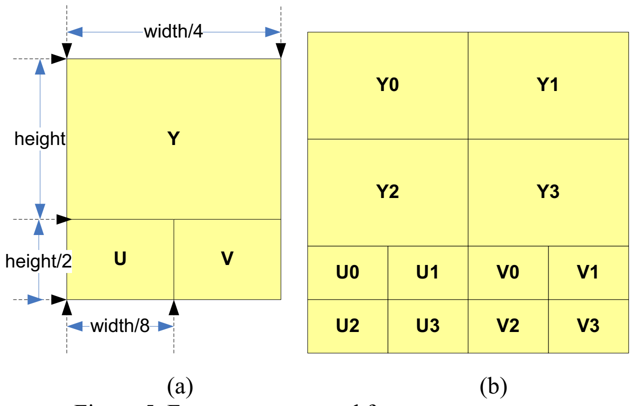


Figure 4. Multi-channel stream control example for 4 channels.

3.2 Frame memory structure

During decoding H.264 stream, we use YUV 4:2:0 format for intermediate video data. For efficient use of SDRAM, we arranged frame data in 2-dimensional arrays as shown in Figure 5 (a). Since we use SDRAM with 32-bit data width, widths are divided by 4. In this frame structure we can easily access 2-dimensional macro blocks, blocks, and sub-blocks. Also, it is efficient for translating from YUV to RGB in video output module. YUV to RGB translation requires 2 rows of Y data and 1 row of U and V data. One row of U and V data can be accessed as one row of data in this frame memory structure.



(a) (b)
Figure 5. Frame structure and frame arrangement

In case of multi-channel decoding, frame data can be arranged as show in Figure 5 (b). Each number means channel number for each video component. By using this arrangement, video output module can easily demultiplex multi-channel video output as if they are a big one. It can reduce DMA register setting cycles and SDRAM initiation cycles.

3.3 Data transfer architecture

Since H.264 decoding requires a lot of 2-dimensional data transfers, we developed a dedicated DMA controller that can transfer 2-dimensional data efficiently. It has a special register for vertical transfer size with normal DMA control registers. It automatically controls both SDRAM controller and buffer memory bus to transfer 2-dimensional data completely.

For efficient data transfers with SDRAM, we made a specialized data transfer architecture with DMA controller and SDRAM controller as shown in Figure 6. The DMA controller has 3 master ports and 4 slave ports. 2 master ports are connected memory controller in each direction for read from and write to SDRAM controller. The other master port controls data bus for transferring data with buffer memories. By using small FIFOs in each master port read from and write to SDRAM can be overlapped so that waiting time for the SDRAM access can be minimized.

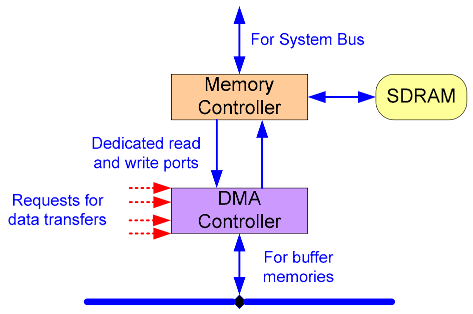


Figure 6. Data transfer architecture.

H.264 decoding has several modes for handling macro blocks and its required transfer block width is different from usual burst sizes. Therefore, we used a special signal to indicate burst size between DMA controller and SDRAM controller. To support arbitrary size of burst transfers we used full-page burst mode and burst termination mode command from SDRAM burst modes.

3.4 Extension to Full-HD resolution

To support various resolutions, we slightly modified SDRAM controller and frame memory related modules. For full-HD decoding without performance degradation, we used 256 Mb SDRAM which has page size of 512 words. It can maintain frame memory structure as shown in Figure 5 (a) within a bank.

4. Experimental Results

We implemented the multi-channel H.264 decoder in Verilog HDL. For fast verification of the design we implemented it with Xilinx FPGA in ChipIt Platinum Prototyping system from ProDesign Gmbh [7] as shown in Figure 7. The host computer sets registers for the number of channels, stream information, frame memory regions and then feeds multiplexed stream to the decoder. We used several well-known video streams to test the design.

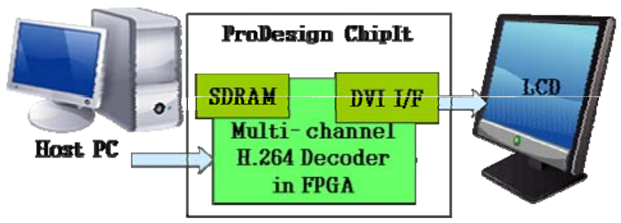


Figure 7. Verification Platform

Figure 8 shows an example output snapshot of 4-channel CIF H.264 decoding with well-known video streams: foreman, coastguard, dancer and singer. To get the multiplexed stream, we encoded each video data and multiplexed them frame by frame inserting channel number.

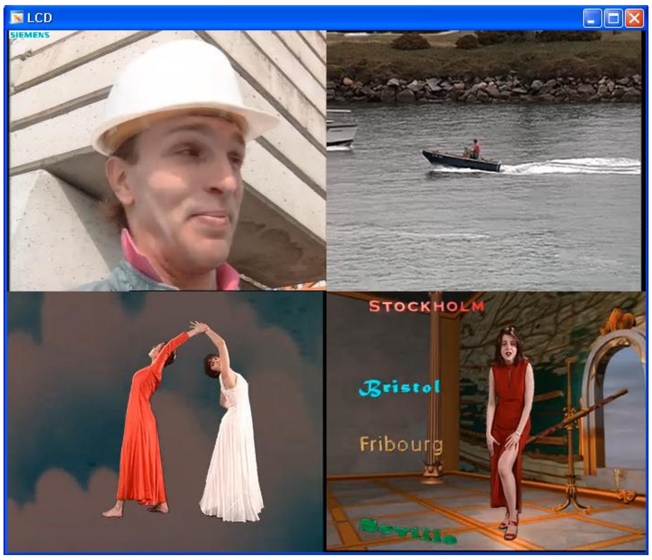


Figure 8. An example output of 4-Channel CIF Decoding.

Table 1 shows the performance results in terms of required clock frequency for decoding 30 f/s of each video type. Four channels of CIF and VGA streams can be decoded in 30 f/s with 28 and 88 MHz, respectively.

Table 1. Performance results for various video types.

Video Type	1-Ch. VGA	4-Ch. CIF	1-Ch. XGA	1-Ch. HD	4-Ch. VGA	1-Ch. Full-HD
Clock [MHz]	22	28	50	66	88	145

It was implemented with 270 K gates and 18 KB of internal buffer memories IBM 90 nm CMOS technology. There were nearly no increase in logic count and the internal memory was reduced by optimizing FIFOs compared to [5]. Power consumption is typically 172 mW for 4-channel VGA 30 f/s at 88 MHz including I/O pads.

5. Conclusions

We extended a single-channel H.264 decoder for multi-channel decoding by efficiently modifying related modules and optimizing critical part of the design. We verified the design with a prototyping system using several well-known video streams. Since it is not a big design, we can incorporate multiple cores for further extension for more than 16 channels of VGA. Later we can apply the multi-channel H.264 to MVC decoder with further modification.

References

[1] AXIS P3301 Fixed Dome Network Camera, Axis Communications, 2008.
[2] "IP Video CCTV and H.264", IndigoVision, http://www.indigovision.com/learnabout-cctvh264.php.
[3] AT2061 Multi-channel H.264 Codec SoC, PentaMicro Inc., http://www.pentamicro.com/, 2007.
[4] ISO/IEC 14496-10 Information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding. 2005.
[5] Wonjong Kim, Seungchul Kim and Hanjin cho, "Autonomous module design methodology for multimedia SoCs", ITC-CSCC, vol. 2, 2007
[6] Krstic, M., et al., System integration by request-driven GALS desig,.IEE Proceedings Computers and Digital Techniques, 2006. 153(5): p. 362-372.
[7] ChipIt Platinum user guide, ProDesign Gmbh, 2007.