A Bus Data Compression Method on a Phase-Based On-Chip Bus

Jaesung Lee

Abstract—**This paper provides a method for compression transmission of on-chip bus data. As the data traffic on on-chip buses is rapidly increasing with enlarged video resolutions, many video processor chips suffer from a lack of bus bandwidth and their IP cores have to wait for a longer time to get a bus grant. In multimedia data such as images and video, the adjacent data signals very often have little or no difference between them. Taking advantage of this point, this paper develops a simple bus data compression method to improve the chip performance and presents its hardware implementation. The method is applied to a Video Codec - 1 (VC-1) decoder chip and reduces the processing time of one macro-block by 13.6% and 10.3% for SD and HD videos, respectively**

Index Terms—**System-on-chip, on-chip bus, multi-media processor, VC-1 codec, video processor**

I. INTRODUCTION

One of the critical issues in the recent multimedia on-chip communications is an enormous increase of data traffic with enlarged video resolutions. This leads to bus contention and makes many IP cores attached to a bus waste time waiting for a grant. However, multimedia signals such as image and video pixel values often have little or no difference between adjacent data. That is, the most significant bits tend to have high spatial and temporal correlations [1]. Taking advantage of this point, IP cores may participate in enhancing chip performance. Namely, they can do some useful tasks such as data compression during the waiting time.

This paper proposes a simple bus data compression method to improve system-on-chip (SoC) performance. Here, 'simple' means it requires no latency in performing the compression and low cost to implement it. The method basically assumes phase-based on-chip buses such as [5, 14] where some guard-bit phase signals exists separately from main channels and indicates current phases of the main channels.

Fig. 1 is a diagram illustrating an example transmission of the compressed bus data according to the scheme this paper proposes. In the figure, 'B' represents a full-byte or 1 byte (8 bits) data, while 'H' represents a half-byte (4 bits) which lacks upper 4 bits. The bus can transmit 2-byte bus data into a slave device in a cycle and nine bus cycles C1 to C9 are described in the figure.

In general, image data are transmitted by 1- or 2-byte unit. In this figure, 1 byte is assumed for the transmission unit. Adjacent (continuous in terms of time) image data

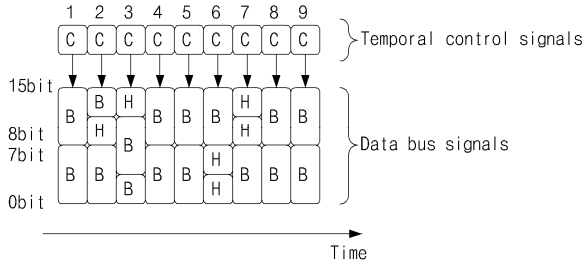


Fig. 1. Diagram illustrating an example bus signal transmission with bus data compression.

Manuscript received Jul. 24, 2011; revised Jan. 9, 2012.
Korea National University of Transportation, 50 Daehak-ro,Chungju-si, Chungbuk 380-702, Korea
E-mail : jaesung.lee@ut.ac.kr

are likely to have the same upper 4 bits. Thus, the data array pattern of each image data is described in two forms of 'B' and 'H' sometimes with the omission of upper 4 bits. Meanwhile, alphabet capital letter 'C' represents a control bus signal. The control signal does not operate at a section where the data bus signal is transmitted in a block. For phase-based buses, these control bus signals are necessary only during initial transmission cycle to deliver various information such as traffic type, burst type and size, and cache control to the slave device. The control bus signals are unnecessary during actual data transmission. That is, upon transmission of the data bus signal, the control signal lines go into a dormant state. In consideration of this point, the signal lines of dormant state are reused to inform a slave device of a pattern of the transmitted data. In general, control bus signals of other standard buses are continuously used during block transfers. This is the reason why the phase-based bus is selected to implement this bus data compression scheme.

An operation of informing the slave device of a compression pattern is performed in each bus cycle. When data is compressed to be transmitted, a slave device is informed of a compressed data array pattern through a related control bus signal 'C'. Hereafter in this paper, the control bus signal that informs about the compressed data array pattern is referred to as a pattern indicator (PI). In this way, low-cost real-time compression of on-chip bus data can be achieved to enhance the performance of multimedia processor SoCs.

The remainder of this paper is structured as follows. Section II briefly reviews the previous work related to the on-chip data compression. Section III illustrates the proposed method and describes its hardware implementation. Section IV evaluates the performance of the method by applying it to a video processor chip that was originally implemented using the de-facto standard bus, Advanced High-performance Bus (AHB). Finally, Section V concludes the paper.

II. Related Work

If some existing data compression algorithms (for example, entropy, code-book, arithmetic, and run-length coding) are applied to bus interfaces, the wasted bandwidth caused by the overlapping transmission of bus data can be reduced significantly. However, in order to implement the data compression algorithms, large-sized buffers, look-up tables, and complex logics are required. This may lead to not only expensive hardware design but also significant transmission delay due to those algorithm operations that could not be parallelized. Consequently, the existing data compression schemes are inappropriate for the on-chip bus data compression.

Studies on the bus encoding schemes (for example, differential coding, gray coding, bus invert coding and the like) have been carried out to reduce bus power consumption [7-13]. The bus encoding schemes encode data values in every bus cycle to reduce the toggling frequency of a signal. However, because the above encoding schemes are focused only on reducing power consumption, the bus bandwidth is still wasted due to transmission of overlapping data.

Meanwhile, there have been extensive efforts to reduce external memory bandwidth, so-called frame memory compression (FMC) [2-4]. But, additional information of each processing unit also has to be stored in internal frame memory; the amount of additional information could detract from the effectiveness of memory compression scheme. S. Hong et al. [15] made a special effort to reduce the limitation. A. Gupte et al. [16] considered power consumption as well as the detraction problem.

However, these approaches focus on the performance enhancement for external memory interface only. Moreover, FMC requires a little bit complex compression algorithms that can introduce some latency.

To address the limitations of those existing efforts, the proposed method employs just a simple coding technique to minimize an increase in cost and make it possible to compress bus data in real time. In addition, the target of the proposed method is not limited to the external memory interface. It includes the interfaces among internal IP cores as well.

III. Bus Data Compression Method and Its Hardware Implementation

In this section, the generalized compression method for bus data in various bandwidth scenarios (8, 16, and 32 bits) is described in detail with reference to Fig.2 through Fig. 4.

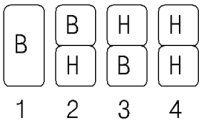


Fig. 2. Data compression patterns on an 8-bit bus.

Fig. 2 is a diagram illustrating compressed data transmission patterns on an 8-bit bus. A leftmost first pattern 1 in the figure includes a first full-byte B (i.e., 1 byte without any omission). A second pattern 2 includes a first half-byte H and lower 4 bits of a second full-byte B. A third pattern 3 includes upper 4 bits of the second full-byte B and a second half-byte H. A fourth pattern 4 includes a third half-byte H and a fourth half-byte H. In this way, the number of all possible data transmission patterns is four in a system having 8-bit bus bandwidth, and the required number of PI signals becomes two (= $\log_2 4$).

Fig. 3 is a diagram illustrating compressed data transmission patterns on a 16-bit bus. The patterns in a 16-bit bus are divided into five types according to the number of half-byte Hs that a pattern has. More specifically, if the number of Hs is zero, as described in Fig. 3, there are two patterns 1 and 2. If the number of Hs is one, there are four patterns 3, 4, 5 and 6. If the number is two, there are three patterns 7, 8 and 9. If the number is three, there are patterns 10 and 11. If the number is four, there is a pattern 12. In this way, the number of all possible patterns becomes twelve of patterns 1 through 12 in a system having 16-bit bus bandwidth. The number of signal lines transmitting a PI signal is four that is a value obtained by applying the ceiling function to $\log_2 12$. But, when lower 4 bits and upper 4 bits are transmitted over two consecutive bus cycles (i.e., the lower 4 bits are firstly transmitted in the antecedent bus cycle among the two bus cycles, and then the upper 4 bits are transmitted in the subsequent bus cycle), the possible patterns in the subsequent bus cycle is limited. For example, patterns that may follow the pattern 2 are limited to the patterns 2, 3, 5, 9 and 10. On the contrary, when the upper 4 bits and the lower 4 bits are not transmitted over two bus cycles, possible subsequent patterns are limited as well. For example, patterns that can succeed the pattern 1 are limited to seven patterns 1, 4, 6, 7, 8, 11 and 12.

Likewise, patterns that can succeed the patterns 1, 3, 5, 7, 8, 10 and 12 are limited to the seven patterns 1, 4, 6, 7, 8, 11 and 12, and patterns that can succeed the patterns 2, 4, 6, 9 and 11 are limited to the five patterns 2, 3, 5, 9 and 10. As a result, just three PI signals are enough to indicate patterns in progress (the ceiling function of $\log_2 7$ is 3).

Fig. 4 is a diagram illustrating compressed data transmission patterns on a 32-bit bus. Data transmission patterns in 32-bit bus bandwidth are divided into eight types according to the number of half-byte Hs. In the case of 32-bit bus bandwidth, the number of necessary signal lines for a PI signal is 7, a value obtained by applying the ceiling function to $\log_2 81$ since there are total 81 patterns.

However, like 16-bit bus bandwidth, the number of patterns in a subsequent bus cycle is limited according to

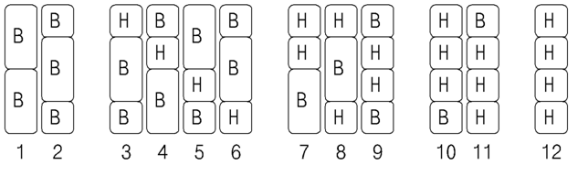


Fig. 3. Data compression patterns on a 16-bit bus.

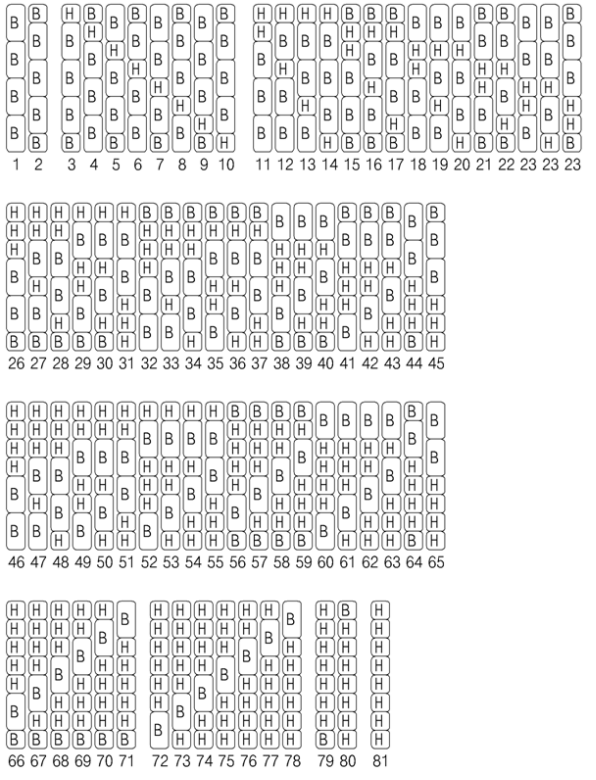


Fig. 4. Data compression patterns on a 32-bit bus.

whether lower 4 bits and upper 4 bits of a full-byte are transmitted over two bus cycles in the 32-bit bus bandwidth. More specifically, patterns that can succeed patterns 1, 3, 5, 7, 9, 11, 12, 13, 14, 18, 19, 20, 23, 24, 26, 27, 28, 29, 30, 31, 38, 39, 40, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 60, 61, 62, 63, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79 and 81 are limited to the patterns 1, 4, 6, 8, 10, 11, 12, 13, 14, 18, 19, 20, 23, 24, 32, 33, 34, 35, 36, 37, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 60, 61, 62, 63, 65, 72, 73, 74, 75, 76, 77, 78, 80 and 81 (total 48). Meanwhile, patterns that can succeed patterns 2, 4, 6, 8, 10, 15, 16, 17, 21, 22, 25, 32, 33, 34, 35, 36, 37, 41, 42, 43, 45, 56, 57, 58, 59, 64 and 80 are limited to the patterns 2, 3, 5, 7, 9, 15, 16, 17, 21, 22, 25, 26, 27, 28, 29, 30, 31, 38, 39, 40, 44, 56, 57, 58, 59, 64, 66, 67 and 68 (total 33). Accordingly, the ceiling function of $\log_2 48$ is 6 and thus the number of necessary signal lines for a PI signal is reduced from 7 to 6.

If the number of signal lines is still insufficient, it is recommended to allow only a part of patterns among the full set of patterns.

All the above illustrations basically assume that their word size is one byte. But, the word size doesn't need to be one byte. For example, if the word size is two bytes, then 'B' becomes two bytes and 'H' becomes one byte where an upper byte is omitted. This information is given to the receiver core in a control phase just ahead of a burst data transmission.

Fig. 5 illustrates an example circuits that implement the bus data compression scheme in hardware. The circuits include a transmitter and a receiver. The transmitter and the receiver are connected to each other through a data bus line and a control bus line. The circuits in the figure transfer compressed bus data in a 16-bit bus bandwidth. Each arrow indicates a flow of a signal, and a numeral indicates a bit width of each signal. The transmitter includes a register, a comparator, and an aligner. The receiver includes a decoder and a duplicator/re-shaper. A 3-bit PI signal is transmitted to the receiver through the control bus line.

HB in the register of the transmitter refers to upper 4 bits of 1 byte and LB refers to lower 4 bits. HB combined with LB constitutes one full byte. The comparator compares upper 4 bits of a previous byte with upper bits of a current byte among the plurality of bytes, and processes a part to be omitted. Except the part omitted by

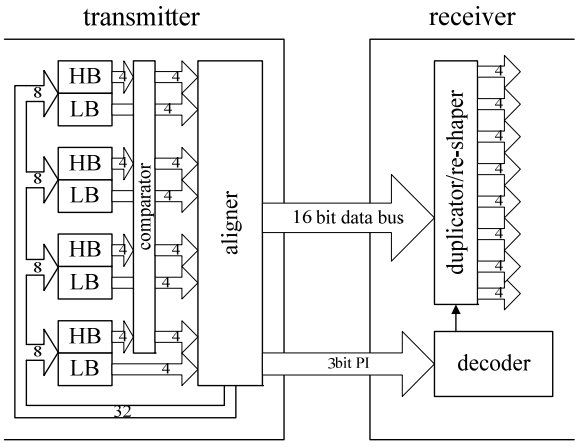


Fig. 5. Data communication circuits transmitting and receiving compressed bus data.

the comparator, remaining data is arrayed on the aligner. The aligner compresses the data in a combination of a full-byte and a half-byte by allowing the current byte to be constituted with the half-byte excluding the upper bits of the current byte if the upper bits of the previous byte are identical to the upper bits of the current byte, and arraying the compressed bus data in a predefined bus bandwidth. The aligner transmits a pattern indicator about a pattern of the compressed bus data to the receiving end.

A receiver device is informed of data array patterns to be sent through a 3-bit PI signal. The PI signal is time-multiplexed with the control bus signal on the inside of the aligner. In the control phase (not data transfer phase), the aligner sends traffic type, burst type, data size, and cash control through the control bus. The multiplexer is not shown to simplify the drawing. Remaining data that do not be arrayed in a corresponding bus cycle is fed back to the register, and re-arrayed in order.

The decoder located at the receiver decodes the PI signal delivered through the control bus lines, determines a shape of a data array, and delivers the information to the duplicator/re-shaper block. The duplicator/re-shaper block restores an omitted HB from continuous byte information and arranges the recovered full byte for use in the next stage.

If the bus bandwidth is 32-bit wide, the aligner may transmit the compressed bus data including one to eight half-bytes in the 32-bit bus bandwidth and the number of lines of the control bus may be six.

IV. PERFORMANCE ESTIMATION

The proposed bus data compression method is applied to a video processor that implements the decoding algorithm of the Video Codec - 1 (VC-1), which is better known as Windows Media Video 9 (WMV 9). Like the MPEG standards, the VC-1 also processes image data by the unit of a macro block of 16x16 pixels, subtracts current pixel values from the pixel values of a reference block determined through motion estimation, and then transforms the residual values. However, there are many differences as well. For example, it uses a DCT-like integer transformation instead of the classic DCT to minimize the computational complexity and perform down to a 4x4 matrix transformation. In addition, one macro block could have four different motion vectors and a motion vector is obtained with a 1/4 pixel-level interpolation of a reference image block. And it uses far more complex de-blocking filtering algorithm.

Originally, the processor was implemented based on a 16-bit version of the AMBA AHB architecture [6] as shown in Fig. 6. A total of nine IP cores are integrated to implement the decoding algorithm. An ARM7 processor core and a DMAC (Direct Memory Access Controller) core control the entire chip as the bus masters. Among the others, those that are directly attached to the bus behave passively as slaves. The core modules for hardware acceleration are variable length decode & inverse quantization (VLD-IQ), inverse DCT (IDCT), motion compensation (MC), and loop filter (LF). These acceleration modules were selectively implemented in hardware, after profiling the VC-1 reference software in detail. There are two types of memories, internal SRAM and external SDRAM, to be used for a frame memory and a line-of-block buffer, respectively. The final two cores, input stream control (ISC) and video output module (VOM), are responsible for interfacing the chip to the outside of it. All the IP cores have 16-bit data interfaces. Some peripherals are also used but not shown in the figure for the sake of convenience.

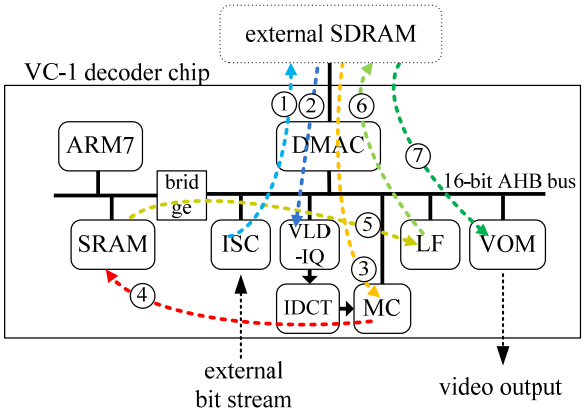


Fig. 6. AHB-based VC-1 decoder.

There are mainly seven data streams (① ~ ⑦) which show up on the bus as the decoding sequence proceeds.

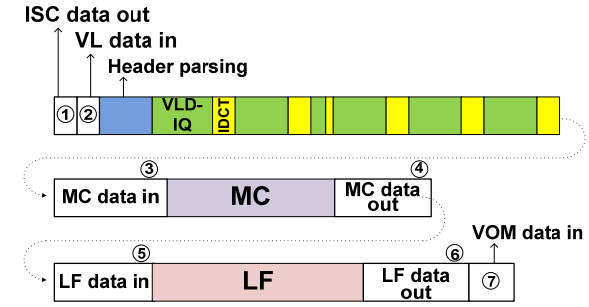
Table 1 summarizes the numbers of clock cycles required for each IP module's operation (in the second row) and its in/out data transfers through the system bus (in the third and fourth rows). Each number is calculated based on one macro block processing for P frames. For I frames, both their occurrence frequency and computational complexity are very lower than those of P frames and thus they are not dealt with.

As shown in the table, the VDL-IQ and IDCT modules have the largest variation in their operation cycles, while the MC and LF modules have the fixed numbers of cycles. In the case of bus use, MC-related bus traffics (③ and ④) and LF-related bus traffics (⑤ and ⑥) occupy the bus for a much longer time than the other modules. Note that all the data transfer cycles in the table include address calculation latency of the DMAC module between back-to-back bock transfers.

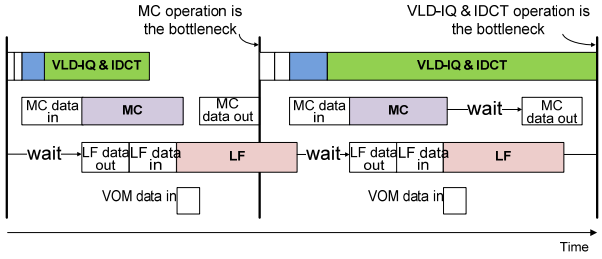
Fig. 7(a) shows the whole operational sequence of the VC-1 decoding, which is roughly drawn proportionally to the numbers of cycles listed in Table 1. This is fully sequentially drawn without consideration of pipelining. The white blocks indicate bus traffics only one of which is allowed to occupy the system bus at any given time. Fig. 7(b) shows the MB-based pipelined operation sequence. The two sets of operations, VDL-IQ & IDCT

Table 1. the number of cycles consumed by each IP module

	ISC	VLD-IQ	IDCT	MC	LF	VOM
Op. cycles	-	317 (all skip) ~ 4026 (4MV)	24 (all skip) ~ 578 (4MV)	1,132	1,451	-
Data-In	-	1 ~ 128 ②	-	693 ~ 848 ③	689 ⑤	362 ⑦
Data-Out	1 ~ 128 ①	-	-	488 ~ 659 ④	663 ~ 807 ⑥	-



(a) sequential (about 9,000 cycles on average)



(b) pipelined (about 5,000 cycles on average)

Fig. 7. Operational flow of IP modules for the VC-1 decoding process.

and MC, are pipelined each other as presented well in the figure, while the LF operation is separated from that pipeline since the SRAM delays the operation by one line of macro blocks, there are also enough time for the LF operation and its in- and out-data transfers, and thus it is not constrained by the pipeline period. Since VLD-IQ & IDCT operation time varies widely from 317 + 24 to 4,026 + 578 cycles while the MC operation time is fixed to 1,132 cycles from Table 1, either VLD-IQ & IDCT or MC operations can be a bottleneck in pipelining.

As shown in the figure, MC-related bus traffics (③ and ④) and LF-related bus traffics (⑤ and ⑥) occupy most part of the bus bandwidth. It is also shown that, when the VLD-IQ & IDCT operation time is extended, the LF module in the next stage should wait for a long time to get a bus grant to move its result data to the SDRAM.

To provide PI signals, the AHB bus is replaced with a 16-bit version of the SoC Network Protocol (SNP) bus [14]. Bus-based interconnects such as AMBA AHB have been widely used for on-chip communication in relatively small SoCs because of simplicity and low cost implementation. However, as the number of intellectual property (IP) cores integrated into a single chip increases, the interconnect architectures based on a single bus becomes unable to sustain the popularity due to their limited bandwidth. Thus, multi-layered buses having mezzanine or piggy-back structures are devised to increase the bandwidth. Recently, bus matrix-based communication architectures are being considered by designers to meet the high bandwidth requirement of modern SoC designs and thus widely studied. However, one drawback of the bus matrix structure is that it connects every master to every slave cluster (or local bus) in the system, resulting in a prohibitively large number of busses. The space occupied by the bus wires, especially high-speed wires, increases dramatically as the number of masters and slave clusters does. The excessive wire congestion can make it practically impossible to route and achieve timing closure for the design.

SNP is devised to reduce the excessive number of bus signals [20]. It is a phase-based communication protocol. In SNP, bus wires are categorized into three groups, control, address, and data signals, and SNP uses only one set of wires called channel (denoted as CHANNEL) to transmit three groups of signals in a time-multiplexed manner. Since the three groups of signals are transferred through the same channel, 3 bits (denoted as PHASE) are used to encode the current phase.

Fig. 8 shows a basic phase-based interconnection between a sender and a receiver. In addition to CHANNEL and PHASE signals, two signals, VALID and READY, are used such that VALID indicates that the channel carries valid information while READY indicates that the receiving agent is ready to latch the incoming information. All signals are directed in the same way from the sender to the receiver except READY directed from the receiver to the sender. Note that many other protocols such as AHB and AXI use the ready and valid signals, too.

Existing SoCs frequently employ master-to-slave communication, such that a master initiates a certain command and then a slave receives and responds to it accordingly. For example, if a microprocessor (master)

Fig. 8. Signals of a phase-based interconnection.

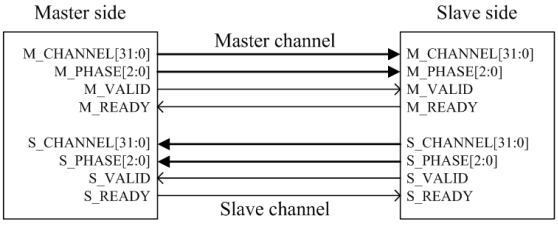


Fig. 9. Master-to-slave connection using SNP channels.

sends an address and command to a memory (slave) to read data, the memory sends the corresponding data. The master-to-slave communication can be implemented using two channels as shown in Fig. 9.

One channel, called the master channel hereafter, includes signals directed from a master to a slave while the other channel, called the slave channel hereafter, is directed from the slave to the master. For the differentiation of the master and slave channels, prefixes M_ and S_ are used for the master channel and the slave channel, respectively. Address, write data and all control phases are carried by M_CHANNEL signals in the master channel. In the slave channel, S_CHANNEL carries read data and response phases. Fig. 9 shows the case when a master and a slave are directly connected to each other. However, multiple masters can be connected to multiple slaves using multiplexers and arbiters. Note that how to build a bus architecture with SNP is explained well in [5].

For AMBA AHB, address and data buses are not used in the same bus cycle; i.e., their information are transmitted sequentially. SNP supports phase omission-restoration to reduce the number of control phases as many as possible. Thus, the time delay of SNP caused by the time-multiplexed transmission is not a problem, when compapred to AHB. In addition, for AHB, the separately defined control bus is used only at the first cycle of the transaction for a burst transfer. As a result, the performance of SNP is almost equal to that of AHB, although SNP has wires that are approximately three-fifths the size of those of AHB.

Moreover, the phase signal can be used for various purposes. The phase signals are unnecessary during actual data transmission. That is, upon transmission of the data bus signal, the phase signal lines go into a dormant state. At that time, the phase lines can be used for a useful purpose; i.e, they are used to inform a slave device of a pattern of the transmitted data. For reference, Intel QuickPath Interconnect or QPI [17] utilize this dormant state to send cyclic redundancy check (CRC) bits through its side-band signals to speeds up the transfer of data packets. It uses only 36 clocks to transfer a data packet including its CRC field as compared to 40 clocks of older architecture. In addition, on-chip bus serialization can benefit from this kind of channel-based approach [19].

Now, the internal buffers for outputting data and their control logics in the MC and LF modules are modified to support the proposed compression method in real time. In addition, real-time decompression logics for data input to the LF module are implemented in the LF module. The SRAM core is regenerated using a memory compiler so that it has 32-bit data in and out interfaces to facilitate the real-time decompression and compression, respectively. Finally, a 32-bit wide block buffer is added to the DMAC for writing data to the SDRAM. It is used on the LF-to-SDRAM data transmission to decompress incoming data promptly. For readers' information, there is no revision for the SDRAM-to-MC data transmission since the effect of the compression scheme is not feasible as reading data from the 16-bit wide external SDRAM does not provide a chance of compression. To estimate only the performance of bus data compression method, some other advantageous features of SNP are not implemented in this modification.

Both versions of the chip, the AHB-based one and the SNP-based one, show the same performance in the VC-1 decoding time when the function of the bus data compression is turned off. However, when it is turned on, the performance is significantly enhanced on average. Fig. 10 shows the 13.6% time reduction of the average pipeline period in which only the MC- and LF-related operations are performed (i.e., the traffics, ①, ②, and ⑦ are not included). The pipeline period is reduced by 11.2% as a whole (i.e., when the traffics, ①, ②, and ⑦ are included).

From Fig. 7, it is found that the LF module has enough time to compress its output data to move them to the SDRAM ⑥ after the LF operation. In addition, Thus, the time for the LF-to-SDRAM data transmission is reduced much as shown in Fig. 10. The SRAM is also able to compress its output data in real time as it has a 32-bit wide data interface, even though it is not given marginal times like the LF's bus waiting time. Thus, the

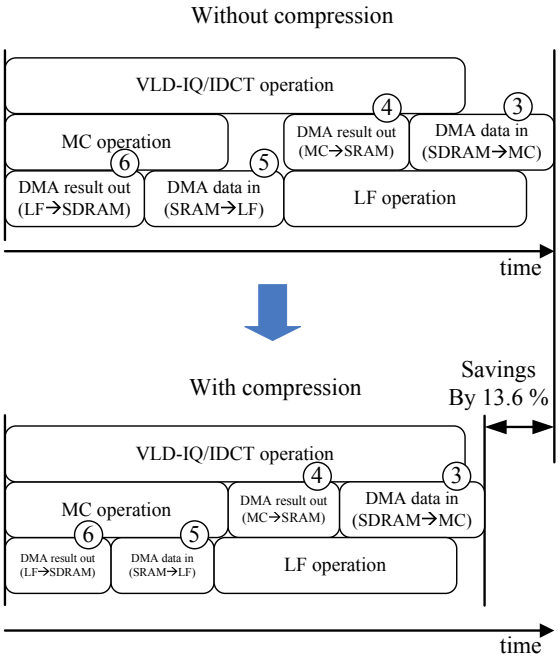


Fig. 10. Reduced pipeline period with the bus data compression.

time for the SRAM-to-LF data transmission is significantly reduced as well.

Meanwhile, the MC module has less marginal time to compress its output data than the LF module when the MC operation is the bottleneck of the pipelining (that is, skip modes frequently occurs). In fact, the VC-1 test bit streams such as Akiyo, Foreman, and Stefan (CIF and 300 frames per each) have 30 ~ 40% of skip modes over all their 8x8 blocks. In addition, as the MC module is scheduled to transfer its result data when the VLD-IQ operations for four luminance blocks are completed (Actually, the appropriate start point for that transmission is hard to estimate), the compression may be not even started when the MC operations for chrominance blocks are skipped. Therefore, the compression effect of the MC-to-SRAM data transmission is relatively minimal as shown in the figure.

Table 2 lists the reduction ratios of the average pipeline periods for the five test videos [18]. Here, it is found that the compression effect is bigger for a motion-abundant video like Stefan and Crowd run.

Table 3 shows the results of logic synthesis using a 0.13 um standard cell library in the Synopsys Design Compiler. The revised version of the VC-1 decoder chip requires a total of 34,755 more gates compared to the original AHB-based version; 12,801 gates acount for

Table 2. Reduction ratios for test videos

Name of video	Reduction ratio
Akiyo (CIF)	6.2%
Foreman (CIF)	14.7%
Stefan (SIF 525)	19.9%
Crowd run (720p)	13.1%
In to tree (720p)	7.5%

Table 3. Increment in gate count

IP module	Original	Revised
VLD-IQ	57,399	-
IDCT	37,830	-
MC	71,311	74,420
LF	51,370	57,542
DMAC	49,385	67,302
SRAM controller	10,548	18,105

replacemet of AHB with SNP (i.e., SNP occupies 12,801 more gates than AHB), and 21,954 gates account for the hardware implementation of the proposed compression and decompression scheme. That is, 34,755 logic gates are paid for the performance enhancement by 13.6% on average for CIF or SIF videos and 10.3% on average for 720p ones.

V. CONCLUSIONS

Recent video applications drive bigger and bigger resolutions. Accordingly, many video processor chips suffer from a lack of bus bandwidth and IP cores residing in those chips have to wait for longer time to get a bus grant. To address the issue, this paper proposes an on-chip bus data compression method. The method utilizes IP cores' waiting times so that they can compress their output data during that time. In addition, its simplicity helps the cores compress data promptly. The method is applied to a video processor and reduces the processing time of one macro-block is reduced by 13.6% for SD videos and 10.3% for HD ones.

Although the results are obtained with SD- or HD-resolution videos since the video processor was developed for a mobile applicaton and thus it supports up to that resoultion, the author thinks that the proposed data bus compression method also takes an enough effect on the large size video such as full-HD videos.

Meanwhile, there may be an opinion that it should be better to use a wider bus rather than this scheme to

enhance system performance. However, the same amount of hardware modification is required to make IP cores support the wider bandwidth. Also, a wider bus might also face design limits such as timing closure and bus area increment caused by a limited number of metal layers.

In addition, assume that a large number of IP cores are integrated into one SoC but only a few IP cores can efficiently support the wider bandwidth. Then, the whole bus replacement can be wasteful and risky due to verification problem. Rather, the author thinks that it is a better solution to modify only the corresponding IP cores to employ the proposed scheme.

ACKNOWLEDGMENTS

The research was supported by a grant from the Academic Research Program of Korea National University of Transportation in 2012.

REFERENCES

[1] P. E. Landman, and J. M. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method," *IEEE Trans. VLSI Syst.,* Vol.3, No.2, pp.173-187, June, 1995.

[2] Y. Jin, Y. Lee, and H.-J. Lee, "A New Frame Memory Compression Algorithm with DPCM and VLC in a 4×4 Block," *EURASIP Journal on Advances in Signal Processing*, Vol.2009, Article ID 629285, 18 pages, Feb., 2010.

[3] T. Y. Lee, "A new frame-recompression algorithm and its hardware design for MPEG-2 video decoders," *IEEE Trans. Circuits Syst. for Video Technology*, Vol.13, No.6, pp.529-534, June, 2003.

[4] Y. V. Ivanov and D. Moloney, "Reference frame compression using embedded reconstruction patterns for H.264/AVC decoders," in *Proceedings of the 3rd International Conference on Digital Telecommunications (ICDT '08)*, Bucharest, Romania, July, 2008.

[5] J. Lee and H.-J. Lee, "Wire Optimization for Multimedia SoC and SiP Designs," *IEEE Trans. Circuits Syst. I*, Vol.55, No.8, pp.2202-2215, Sept., 2008.

[6] J.-H. Kim, "A Hybrid Pipelined Implementation of a Standard Video Decoder with an Aggressive MC Data Reuse," M. S. Thesis, Seoul National University, Seoul, Korea, 2007.

[7] M. R. Stan and W. P. Burleson, "Bus-invert coding for low power I/O," *IEEE Trans. VLSI Syst*., Vol.3, pp.49-58, Mar., 1995.

[8] P. Panda and N. Dutt, "Reducing address bus transitions for low power memory mapping," in *Proc. Int. Symp. Design Automation Test Eur*., pp.63-67, 1996.

[9] L. Benini, G. De Micheli, E. Macii, et al., "Asymptotic zero-transition activity encoding for address buses in low-power microprocessor-based systems," *in Proc. 7th Great Lakes Symp. VLSI*, pp.77-82, 1997.

[10] S. Ramprasad, N. Shanbhag, and I. Hajj, "A coding framework for low power address and data busses," *in IEEE Trans. VLSI Syst*., Vol.7, pp.212-221, June, 1999.

[11] Y. Aghaghiri, F. Fallah, and M. Pedram, "Irredundant address bus encoding for low power," *in Proc. Int. Symp. Low-Power Electron. Design*, 2001, pp.182-187.

[12] L. Macchiarulo, E. Macii, and M. Poncino, "Low-energy for deep-submicron address buses," *in Proc. Int. Symp. Low-Power Electron. Design*, pp.176-181, 2001.

[13] R. R. Rao, H. S. Deogun, D. Blaauw, et al., "Bus Encoding for Total Power Reduction Using a Leakage-Aware Buffer Configuration," *IEEE Trans. VLSI Syst.*, Vol.13, No.12, pp.1376-1383, Dec., 2005.

[14] J. Lee, H.-J. Lee, and C. Lee, "A Phase-Based Approach for On-Chip Bus Architecture Optimization," *The Computer Journal*, Vol.52, No.6, pp.626-645, Aug., 2009.

[15] S. Hong, D. Chung, and Y. Choe, "Selective merging-based reference frame memory compression," in *IEEE Pacific Rim Conf. on Communications, Comuters, and Signal Processing*, pp.896-900, 2011.

[16] A. Gupte, B. Amrutur, M. Mehendale, A. V. Rao, M. Budagavi, "Memory Bandwidth and Power Reduction Using Lossy Reference Frame Compression in Video Encoding," *IEEE Trans. Circuits Syst. Video Techn*., Vol.21, pp.225-230, Feb., 2011.

[17] An Introduction to the Intel QuickPath Interconnect, Intel Corporation, Jan. 30, 2009, Retrieved June 14, 2011. [Online] Available: http://www.intel.com/technology/quick path/introduction.pdf.

[18] Collection of test sequences and clips for evaluating compression technology, xiph.org. [Online] Available: http://media.xiph.org/video/derf/

[19] J. Lee, "On-Chip Bus Serialization Method for Low-Power Communications," *ETRI Journal*, Vol.32, No.4, pp.540-547, Aug., 2010.

[20] B.-G. Ahn, J. Kim, W. Li, and J.-W. Chong, "Effective Estimation Method of Routing Congestion at Floorplan Stage for 3D ICs," *Journal of Semiconductor Technology and Science*, Vol.11, No.4, pp.344-350, Dec., 2011.

Jaesung Lee received the B.S. and M.S. degrees in Electronic Engineering from Ajou University, Suwon, Korea in 1999 and 2001, respectively, and the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea, in 2008. From 2001 to 2011, he was with the Cloud Computing Research Department of Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. In 2011, he joined the Department of Electronic Engineering at Korea National University of Transportation, Korea, where he is currently working as an Assistant Professor. His research interests include multimedia SoC design and computer architecture. He has published more than 30 papers as the first author in international and domestic journals and proceedings and has held more than 10 overseas and domestic patents. He won the best paper award at the 8th Samsung Human-tech Thesis Prize in 2002. Prof. Lee is a member of the IEEE, the IEEK, the KIISE, and the KICS.