On-Chip Bus Serialization Method for Low-Power Communications

Jaesung Lee

One of the critical issues in on-chip serial communications is increased power consumption. In general, serial communications tend to dissipate more energy than parallel communications due to bit multiplexing. This paper proposes a low-power bus serialization method. This encodes bus signals prior to serialization so that they are converted into signals that do not greatly increase in transition frequency when serialized. It significantly reduces the frequency by making the best use of word-to-word and bit-by-bit correlations presented in original parallel signals. The method is applied to the revision of an MPEG-4 processor, and the simulation results show that the proposed method surpasses the existing one. In addition, it is cost-effective when implemented as a hardware circuit since its algorithm is very simple.

Keywords: Low-power design, multimedia, on-chip bus, system-on-chip, bus serialization.

I. Introduction

Today, a fundamental shift is occurring in the system-on-chip (SoC) industry – a shift from parallel I/O schemes to serial I/O interconnect solutions. This change is being driven by many researchers as a means to reduce design costs, simplify system complexity, and provide the scalability needed to meet new bandwidth requirements [1]-[3]. This shift will not be without engineering challenges. One of the biggest challenges is the increased power consumption in on-chip serial communications.

Most on-chip bus communication protocols [4]-[12] adopt a communication process that transmits destination address information of transmit/receive (TX/RX) data in the first bus cycle and transmits/receives the TX/RX data in the next bus cycle. This communication process is performed in a register pipeline manner. Figure 1 shows a diagram illustrating the process of transmitting data on a microprocessor bus. In general, an address bus transmits successive address information. Thus, there are many correlations between the addresses of respective cycles. The address information often increases (or decreases) simply as illustrated in Fig. 1. In the figure, 'A' denotes an overlap between the respective addresses. In this manner, the signals of an address bus are small in signal transition frequency (that is, the number of signal transitions). However, if the address bus signals are serialized prior to transmission, their correlations are very likely to be broken. If the address signals are serialized with their correlations broken, their signal transition increases dramatically. This means that serial communication dissipates significantly more energy than parallel communication.

This is also true in the case of a data bus. Referring to Fig. 1, data bus signals, including data signals Data0, Data1, Data2,

Manuscript received Aug. 2, 2009; revised May 9, 2010; accepted May 25, 2010.

This work was supported by the IT future technology development program (2007-S-016-02) of Ministry of Knowledge Economy (MKE) and Institute of Information Technology Assessment (IITA), Rep. of Korea.

Jaesung Lee (phone: +82 42 860 5762, email: jaesung.lee@etri.re.kr) is with the Software Research Laboratory, ETRI, Daejeon, Rep. of Korea.

doi:10.4218/etrij.10.0109.0447

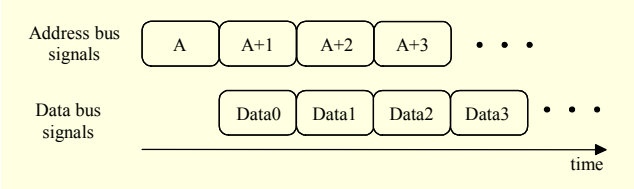

Fig. 1. Diagram illustrating the process of transmitting data on a microprocessor bus.

and Data3, which correspond respectively to address signals A, A+1, A+2, and A+3, generally have a correlation between them, just like address bus signals. For example, in the case of multimedia data, such as in an image, audio, or video, the adjacent data signals very often have little or no difference between them. In common multimedia applications, the most significant bits tend to have high spatial and temporal correlations because of the sign extension or the locality characteristics of multimedia streams [13]. However, like address signals, if multimedia data signals are serialized prior to transmission, their correlations are broken, and accordingly their signal transition increases significantly.

What is therefore required when serializing the parallel bus signals prior to transmission is a method for preventing a sudden increase in the signal transition frequency in order to suppress an increase in power consumption. In response to this requirement, this paper proposes a low-power bus serialization method. This method significantly reduces the frequency by making the best use of the previously mentioned correlations, and employs just a few simple coding techniques to minimize an increase in cost.

The remainder of this paper is structured as follows. Section II briefly reviews the previous work related to low-power bus coding. Section III illustrates the proposed method and describes in detail the complete encoding and decoding algorithms and their hardware implementation. Section IV evaluates the performance of the method by applying it to a MPEG-4 chip that was already implemented using the de-facto standard bus, Advanced High-performance Bus (AHB). Finally, section V concludes the paper.

II. Related Work

There have been many techniques used to reduce the activity factor on a multi-bit bus, that is, a parallel bus [14]-[20]. However, they are not applicable to serial interconnects since the principle of increasing the activity factor in serial communications is different from that in parallel communications as presented in the previous section. The authors in [21] proposed for the first time a low-energy transmission coding method for on-chip serial communications to reduce the activity factor of a serial wire. Currently, this paper is unique in its study of low-energy coding for on-chip serial communications. The technique proposed in [21] reduces the number of transitions on the serial data line in order to lower the energy consumption by saving the data correlation between successive data words. However, the technique misses some useful points and is therefore less optimized even though the activity factor can be further reduced. The next section illustrates that issue in detail and proposes the missing steps that are added to the algorithm used in the technique.

III. On-Chip Bus Coding Method for Low-Power Serial Communications

This section describes in detail the coding method of multi-bit parallel bus signals for low power serial communications. The method encodes bus signals prior to serialization so that they are converted into signals that do not greatly increase in signal transition frequency when serialized. In addition, the conversion does not require communication delay.

Figure 2 illustrates a serialization example for some bus signals. In the figure, '*t*' to '*t*+4' denote a sequence of one-byte words representing five addresses or data that are transmitted successively. In this example, the word size is of a byte, but it can be larger in other situations. Also in the figure, 'b0' to 'b7' denote the bit positions of each byte. Here, a numeral following 'b' denotes the position of each bit. Starting from the first word at time *t*, hexadecimal data of 51h, 52h, 53h, 54h, and 55h are expressed as binary numbers, and they are supposed to be sequentially transmitted through an 8-bit bus. As can be seen on the left side of the figure, the total of signal transitions is 7 (that is, 4 times at 'b0', 2 times at 'b1', and 1 time at 'b2') if the bus signals are transmitted without serialization, but the total of signal transitions is 31 if the bus signals are serialized and transmitted.

With the exception of the first word at time *t*, Fig. 3 illustrates the same example after the other words *t*+1, *t*+2, *t*+3, and *t*+4 are XOR-operated in a bitwise manner. The XOR operation is performed between the previous byte and the current byte in

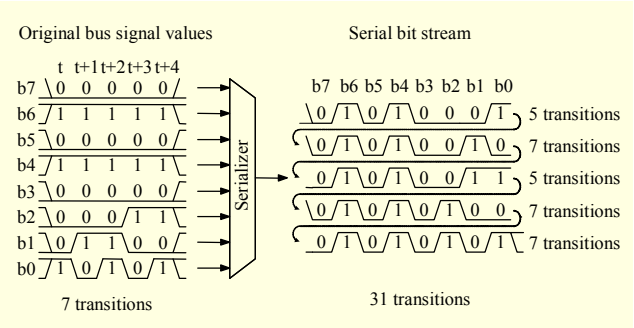

Fig. 2. Serialization example for parallel bus signals.

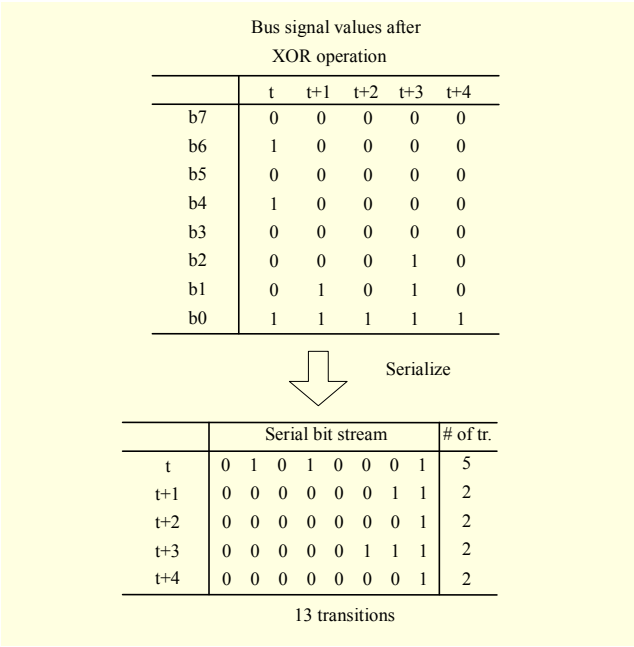

Fig. 3. Serialization example after an XOR-operation is performed.

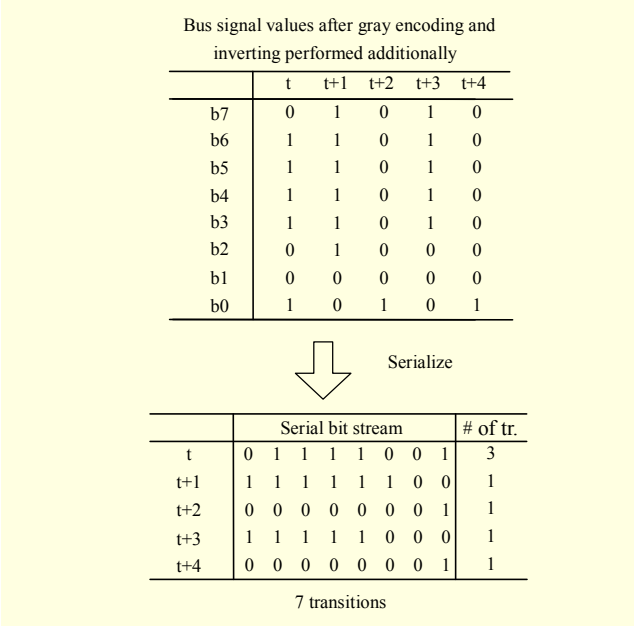

Fig. 4. Serialization example after gray encoding and inverting are performed additionally.

the original signal sequences. This method was proposed in [21]. As shown on the right side of the figure, the number of signal transitions decreases to 13 with this method.

Although the XOR operation significantly reduces the number of transitions, there still remains a chance for the number to be reduced even more as the transition count in the first word is not controlled, and the correlation in word-to-word conjunctions is not considered.

In general, gray encoding can reduce the number of bit-by-bit transitions when it is applied to a bit-serial sequence which has many bit-by-bit transitions. Thus, if the transition count of the original bit sequence is bigger than that of the gray-coded sequence, it would be advantageous to select the latter. Theoretically, the maximum transition count is n–1 for an n-bit data word, but it can be reduced to $\lceil n/2 \rceil$ when the selection is made. Meanwhile, when the most significant bits remain the same while the least significant bits frequently change in a sequence of data words, the bitwise-XOR operation between two adjacent words is likely to make the most significant bits zeros and least significant bits ones. As a result, it significantly increases the probability that a transition takes place in a word-to-word boundary when the words are serialized successively. In this case, the bit inversion of one word can be helpful to remove the transition. Thus, two additional steps are proposed as follows for a further reduction of the frequency of transition.

Figure 4 illustrates the example of serialization after additionally performing gray encoding on the first word and inverting the even-numbered words (that is, t+1 and t+3). The gray encoding XOR-operates the corresponding bit of a binary value that is to be encoded and the previous bit in that bit sequence. The encoder works as follows:

$$B[n-1] = b[n-1],$$

$$B[i-1] = b[i] \oplus b[i-1], \quad \text{for } i = n-1 \sim 1 \ . \tag{1}$$

Here, b[n–1:0] represents an n-bit data word and B[n–1:0] represents an n-bit gray-encoded data word. Note that the first bit of the binary value is not operated and reflected into the first bit position of the gray code word as it is. As illustrated in Fig. 4, the frequency of signal transition of the first word at time t decreases to 3. In addition, with the exception of the first word at time t, inversion of the even-numbered words makes each of the other byte sequences t+1, t+2, t+3, and t+4 have its transition count reduced by one. As a result, the total signal transition frequency decreases down to 7. This is because the additional steps of the operation make it possible to take advantage of the correlation in word-to-word conjunctions and includes the first word in the encoding process as well. Note that the transition count of 7 is the same as the transition count on the original parallel bus as shown in Fig. 2.

1. Encoding Algorithm and Its Hardware Implementation

Figure 5(a) shows a flow chart illustrating the complete algorithm of the bus signal encoding method. In step 1, the algorithm takes a stream of parallel bus signals, and in step 2, it XOR-operates all but the first word in the signal stream, word by word, in a bitwise manner as illustrated previously. Then, it inverts the even-numbered words in a bitwise manner (step 3).

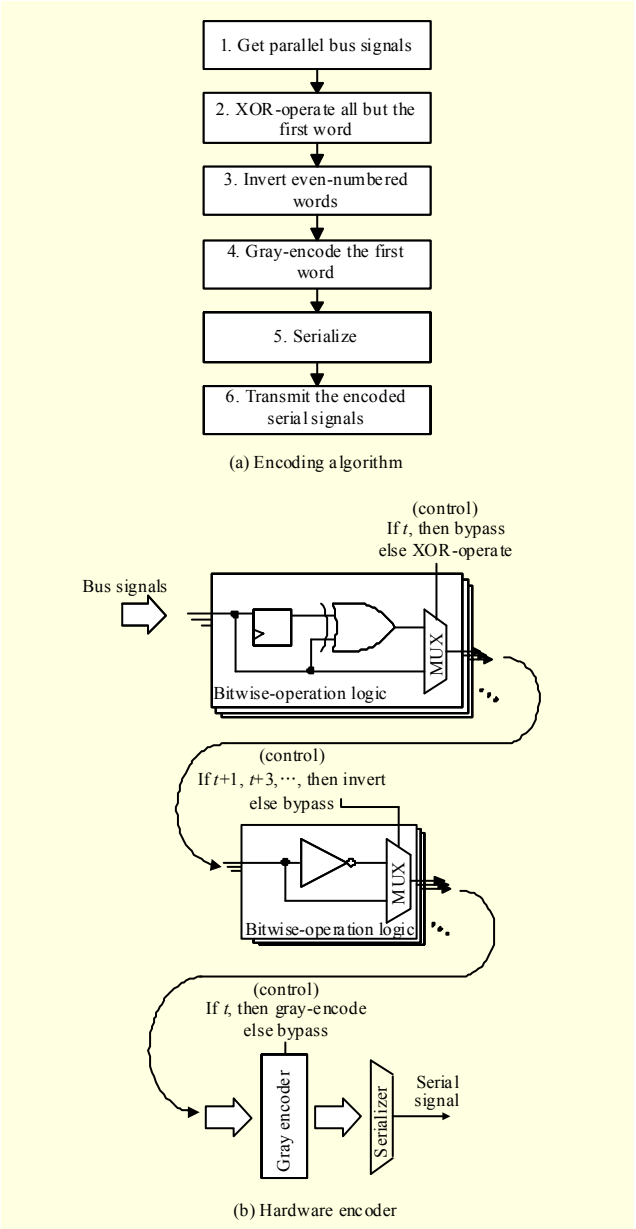

Fig. 5. (a) Bus signal encoding algorithm and (b) its hardware implementation.

In step 4, it performs an operation of gray-encoding on the first word, depending on the bit-by-bit transition count of the first word of the original signal (that is, only if the count is bigger than that of the gray-coded signal is this operation performed). Then, the bus signal encoding method serializes the modified parallel bus signals in step 5 and transmits the serial signals to a destination device in step 6. Note that the gray-encoding step can be placed anywhere between receiving the bus signals (step 1) and serialization (step 5) since it is orthogonal to the other steps of operation.

Figure 5(b) shows a block diagram of a hardware encoder that implements the low-power bus serialization algorithm. It consists of an XOR operation block, an inverting operation block, a gray encoder, and a serializer. The XOR operation block latches each bit of a parallel bus signal and XOR-operates the stored bit in the previous cycle and the current incoming bit. The inverting operation block inverts the XOR-operated parallel bus signals except the first word in a bitwise manner. Here, the block inverts only the even-numbered words of the parallel bus signals in a bitwise manner. For the odd-numbered words, it just bypasses them. The first word excluded from the XOR operation is gray-encoded in the gray encoder. Finally, the serializer transforms the modified parallel bus signals into serial wire signals. All the logic circuits perform their operations clock by clock in a pipeline manner.

The gray encoder may be implemented by using a lookup table (LUT) storing the operation results, or by using a so-called XOR chain that uses the first bit value as the operation result for the first bit, and XOR-operates the current bit with the previous bit from the second bit.

2. Decoding Algorithm and Its Hardware Implementation

Figure 6(a) shows a flow chart illustrating the complete algorithm of the serial wire signal decoding method. In step 1, the algorithm receives a stream of serial bit signals. In step 2, it deserializes the received serial bit signals into parallel bus signals. In step 3, the first word of the parallel signals is gray-decoded or not, depending on the value of the indicator signal, which is a guard bit signal generally existing for the control of the serial address or data transmission [21] or a bit signal transmitted at the very beginning in front of the regular transmission of the serial information. The gray-decoding method is exactly the same as the gray-encoding method. It then inverts the even-numbered words of the deserialized parallel signals in a bitwise manner in step 4 and XOR-operates all but the first word in a bitwise manner to restore the original bus signals in step 5. Finally, the restored parallel signals are consumed inside the receiving circuit (step 6).

Figure 6(b) shows a block diagram of a hardware decoder that implements the decoding algorithm. Similar to the encoder, it consists of a deserializer, gray decoder, inverter operation block, and XOR operation block. However, the series of operations are performed in reverse order of that for the encoder. Except for the deserializer, each operation of the block is the same as in the encoder.

In parallel bus communications, address and data information is transmitted in a unit of a certain size (for example, an 8-bit bus, 16-bit bus, 32-bit bus, and so on). Hence, the receiving circuit can detect the boundary of each word even between successive serial data (or address) transmissions as well as the first base signal value (that is, the first word) of each

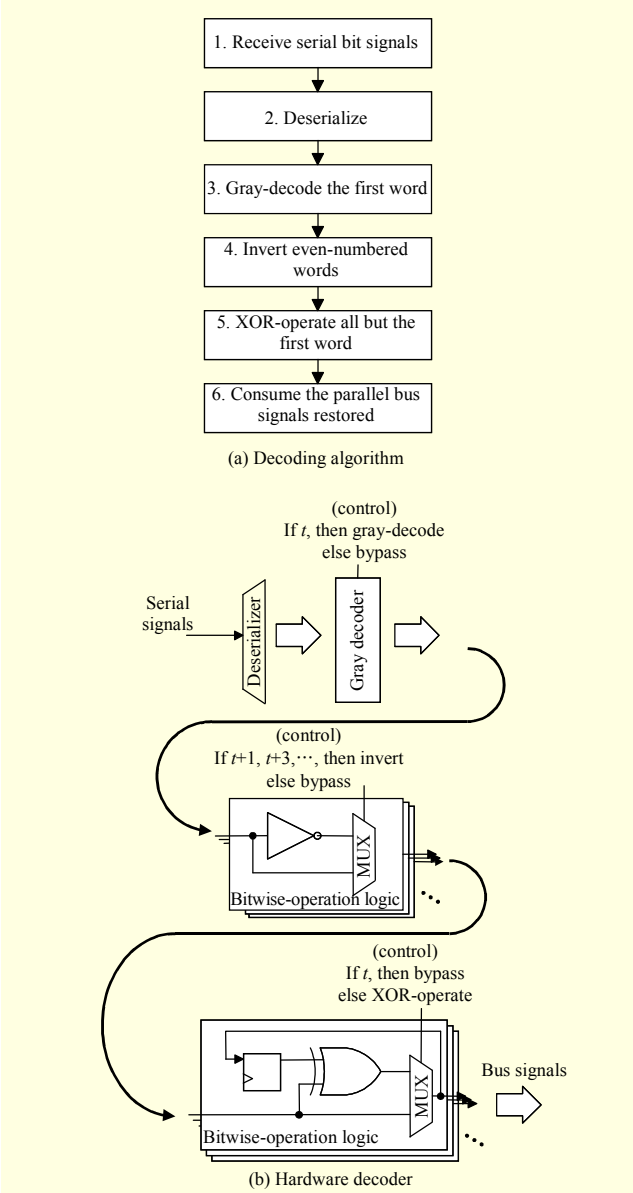

Fig. 6. (a) Bus signal decoding algorithm and (b) its hardware implementation.

transaction. As a result, there is no problem in performing those operations in the decoding process. In addition, the gray decoder can be implemented using an LUT or XOR chain because the gray decoder is exactly the same as the gray encoder.

From Figs. 5(b), 6(b), and (1), it can be inferred that the increased logic area for the encoder and decoder circuits is very small (actually, only 1,769 gates are used on average when synthesized). Thus, the additional cost is ignorable in that gate counts of today's ICs range from a few to a few tens of millions.

Meanwhile, the algorithms introduced in this paper do not require any clock cycle delay as the implementation of their operations requires combinational logics only. In addition, all the operations are performed in parallel not serially. Only the serialization in the 5th stage of Fig. 5(a) and the deserialization in the 2nd stage of Fig. 6(a) require high speed clocks and they are implemented using designated serdes cells in a cell library. Thus, the circuits that implement the algorithms will not degrade the performance of the whole chip.

IV. Performance Estimation

The proposed bus serialization method is applied to the revision of an MPEG-4 processor [22] that was already implemented based on the existing on-chip bus, Advanced Microcontroller Bus Architecture (AMBA) AHB. Figure 7 shows the communication architecture on the inside of the processor. Thirteen hardware modules are used to implement an MPEG-4 algorithm. Among these modules, the ARM7 core controls the entire chip and Direct Memory Access Controller/SDRAM Controller (DMAC/SDRAMC) interfaces with external SDRAM. Three hardware modules, input stream control (ISC), video input control (VIM), and video output control (VOM), also have interfaces with the outside of the chip. Other modules are designed for the acceleration of complex computation. The hardware acceleration modules are motion estimation coarse (ME-C), motion estimation fine/motion compensation (ME-F/MC), discrete cosine transform and quantization/inverse quantization and inverse discrete cosine transform (DCTQ/IQIDCT), variable length decoding (VLD), macro-block reconstruction (REC), image de-blocking (DB), variable length coding (VLC), and stream producer (SP) modules. All hardware modules are connected through the parallel signal lines of the 16-bit version of AHB as shown in the figure. Note that the 16-bit bus signals are processed in a unit of a byte on the coding stages for serialization in the revised version of the processor. The details of the original implementation of the processor are well presented in [22].

Wrappers are used to interface the hardware modules to the bus. Inside the wrappers, 'Master' and 'Slave' denote a master-type interface and a slave-type interface, respectively. In the bottom of the figure, there are also several back-buses due to the insufficient bandwidth of the AHB. Wrappers denoted by 'I/F' are used to interface the associated modules to the back-buses.

To apply the serialization method to this processor, all the wrappers are equipped with the encoder and decoder circuits presented in the previous section, and all the buses are replaced with full-duplex serial wires. Inside the wrappers, serial data transceiver cells are also included to handle the physical signaling. For the back-buses, the unidirectional serial wires are

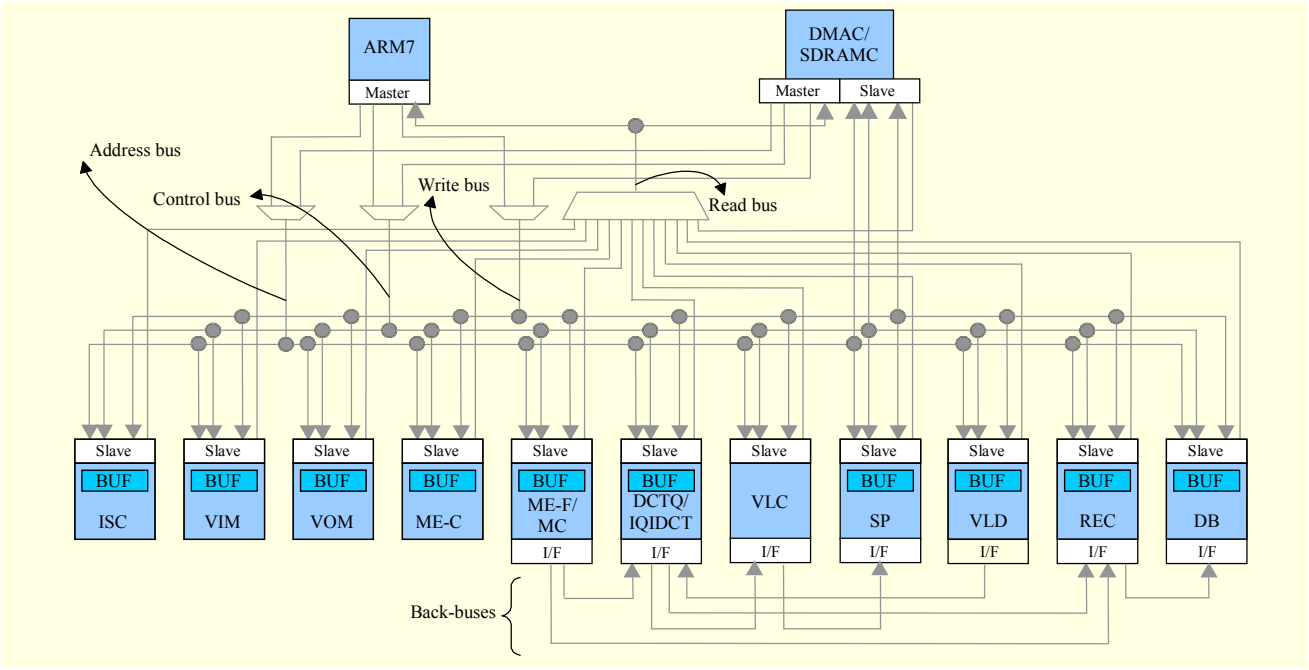

Fig. 7. Communication architecture of MPEG-4 processor.

used instead of full-duplex wires since the associated modules have one-way traffic only. As most of the wrapper interface wires are bussed together with those of many other wrappers, a transition on a wrapper interface is very likely to propagate throughout the entire communication path.

Figure 8 shows the simulation results of an MPEG-4 compression for two sample video clips, Foreman QCIF and Tennis QCIF, that were successively processed. Each bar in the graph presents the relative transition count on serial wires attached to the corresponding module's wrapper, as the transition counts associated with the AHB-based version (denoted by 'Parallel' in the figure) are used as a reference. Therefore, the bar values for the AHB-base version are all equal to one. Here, the relative value is used because an absolute value increases dramatically so that it is clumsy to present the absolute value with a number of digits. In the case of direct serialization of the AHB signals without any coding operations enabled, the bars denoted by 'non-coding' in the graph present the relative counts for the revised version of the processor. The bars denoted by 'XOR' represent the relative counts when only the XOR operation is enabled as in [21]. The bars denoted by 'Complete' represent them when all of the three operations in the algorithm are enabled. The results for the modules, ISC, VLD, and DB, are not shown in this graph because they are not used in the compression process. They are used only in the decompression process.

As shown in the graph, the transition increase through the serialization without any coding operations (that is, non-coding) reaches from about three to four times the transition counts of the parallel bus, (that is, AHB). However, the transition increase is significantly reduced with the coding operations. Among the modules, the VIM and VOM modules benefit the most from the bus coding because they mostly handle raw image data, and thus there are very large correlations between consecutive data in most cases. Meanwhile, for the VLC and SP modules, the benefit is minimal because they mainly handle entropy-coded words. On average, the XOR operation leads to about 30% energy saving as seen at the bottom of the graph. With all of the three operations enabled, the saving is about 20% higher than that of the XOR operation. The proposed coding method shows an enhanced performance for all of the modules.

Although the results in the graph include the transitions to transmit address information as well, since most of the modules communicate with other modules in a unit of a macro block of image data through their internal buffers, the addressing patterns are merely a simple increment or simple decrement, and thus their effects on the transition counts are minimal.

Figure 9 shows the simulation results of MPEG-4 decompression for the two sample video clips. The graph conventions are identical to those for Fig. 8. The results for the modules, VIM, ME-C, VLC, and SP, are not shown in this graph because they are not used in the decompression process.

The transition increase through the serialization is similar to that in the compression process. All of the modules benefit from the coding operations. Among the modules, the VOM and DB modules benefit the most from the bus coding because

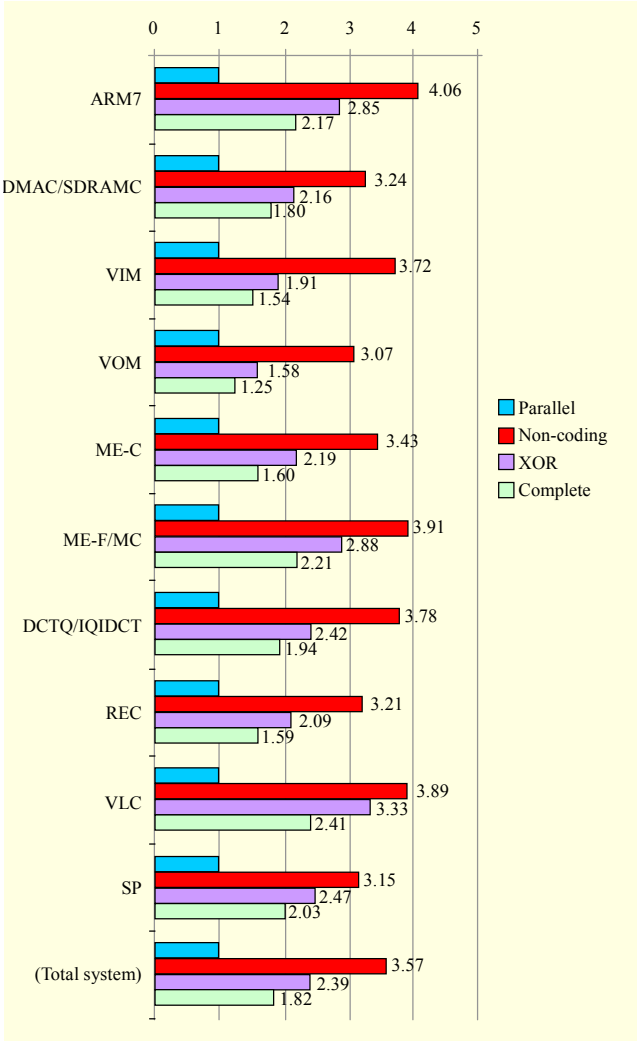

Fig. 8. Simulation results of MPEG-4 compression.

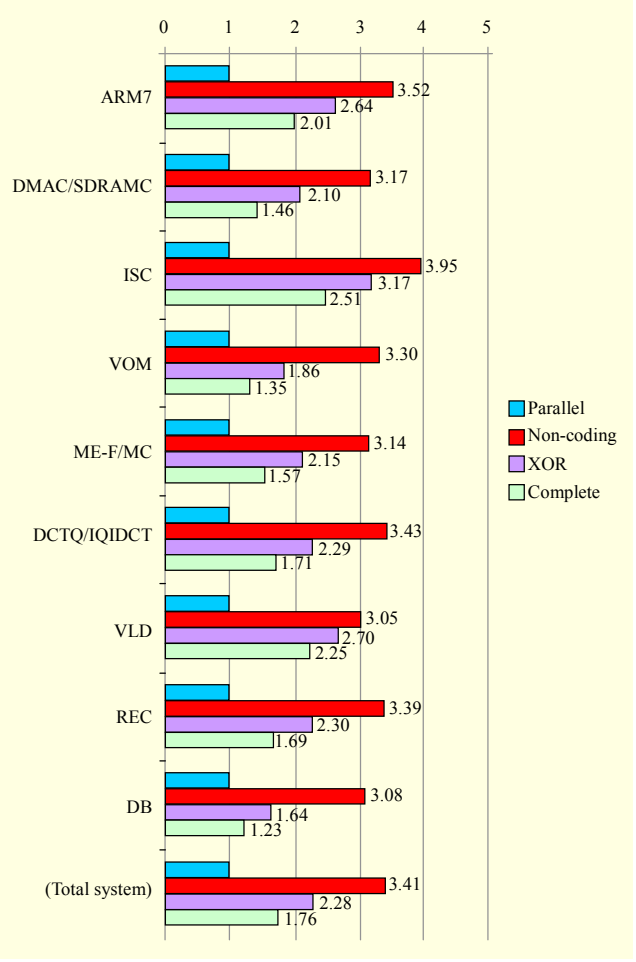

Fig. 9. Simulation results of MPEG-4 decompression.

they mostly handle raw image data. Meanwhile, for the ISC and VLD modules, the benefit is minimal because they mainly handle entropy-coded words. Likewise, enabling the three operations saves about 50% in energy compared with the direct serialization without any coding operations.

A signal transmission path or a conductive line external to an on-chip core module has hundreds to thousands of times the capacitance of a conductive line on the inside of the core. Therefore, the former consume hundreds to thousands of times more power per signal transition than the latter [21]. In this evaluation, the transition count values in the graphs include those within the codec circuits, reflecting the conversion factor.

The total gate count of the codec circuits, that is, the hardware encoders and decoders, is 23k. However, the additional cost is ignorable in that the total gate count of the chip is about 2M gates.

To investigate the overall power consumption of the MPEG-4 processors, the power of the two processors is estimated using Synopsys DesignPower. One processor includes the encoding and decoding scheme and one does not. Two sets of synthesized netlists, a power library (Samsung 0.13 μm) which contains power information of gates that compose the netlists, and switching activity interchange files that are obtained from the above RTL simulation. Then, DesignPower is run with them.

As a result, the overall power consumption of the MPEG-4 processor that does not include the encoding and decoding scheme is 673 mW. The overall power consumption of the MPEG-4 processor that adopts our encoding and decoding scheme is 602 mW. That is, 10.5% of the power is saved. Although the power saving in the total chip level is minimal considering the transition count reduction 50% in the bus level, the scheme is still meaningful. As the battery life of today's hand-held devices becomes longer, the lifetimes of those devices will be extended by the saved power proportionally.

V. Conclusion

To address the low power issue in a serialized transmission

of on-chip bus signals, this paper proposes a method of suppressing the increase in signal transition counts by saving the word-to-word and bit-by-bit correlations as much as possible. The main contribution of this paper is to devise two steps of operations that are added to the existing algorithm and evaluate the performance of the algorithm by applying it to a revision of a commercial SoC MPEG-4 processor. From the simulation results, it is shown that the proposed method surpasses the existing one. In addition, a low cost implementation of the algorithm is possible without communication delay as the algorithm is very simple.

References

[1] S.J. Lee et al., "An 800 MHz Star-Connected On-Chip Network for Application to Systems on a Chip," *ISSCC Dig. Tech. Papers*, 2003, pp. 468-469.

[2] S. Kimura et al., "An On-Chip High Speed Serial Communication Method Based on Independent Ring Oscillators," *ISSCC Dig. Tech. Papers*, 2003, pp. 390-391.

[3] K. Lee et al., "A 51 mW 1.6 GHz Network for Low-Power Heterogeneous SoC Platform," *ISSCC Dig. Tech. Papers*, 2004, pp. 152-153.

[4] ARM, "AMBA Specification, Rev. 2.0," 1999.

[5] IBM, "CoreConnect Bus Architecture," 1999.

[6] SuperH, "SuperHyway On-Chip Interconnect Solution," 2003.

[7] Palmchip, "CoreFrame Architecture," 2002.

[8] STMicroelectronics, "STBus Interconnect," 2004.

[9] OpenCores, "Wishbone bus," 2003.

[10] ARM, "AMBA AXI Protocol Specification," 2003.

[11] VSI Alliance, "Virtual Component Interface Standard," 2000.

[12] OCP International Partnership, "Open Core Protocol Specification," 2001.

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

[14] M.R. Stan and W.P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Trans. VLSI Syst.*, vol. 3, no. 1, Mar. 1995, pp. 49-58.

[15] P. Panda and N. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping," *Proc. Int. Symp. Design Automation Test Eur.*, 1996, pp. 63-67.

[16] L. Benini et al., "Asymptotic Zero-Transition Activity Encoding for Address Buses in Low-Power Microprocessor-Based Systems," *Proc. 7th Great Lakes Symp. VLSI*, 1997, pp. 77-82.

[17] S. Ramprasad, N. Shanbhag, and I. Hajj, "A Coding Framework for Low-Power Address and Data Busses," *IEEE Trans. VLSI Syst.*, vol. 7, no. 2, June 1999, pp. 212-221.

[18] Y. Aghaghiri, F. Fallah, and M. Pedram, "Irredundant Address Bus Encoding for Low Power," *Proc. Int. Symp. Low-Power Electron. Design*, 2001, pp. 182-187.

[19] L. Macchiarulo, E. Macii, and M. Poncino, "Low-Energy for Deep-Submicron Address Buses," *Proc. Int. Symp. Low-Power Electron. Design*, 2001, pp. 176-181.

[20] R.R. Rao et al., "Bus Encoding for Total Power Reduction Using a Leakage-Aware Buffer Configuration," *IEEE Trans. VLSI Syst.*, vol. 13, no. 12, Dec. 2005, pp. 1376-1383.

[21] K.M. Lee, S.J. Lee, and H.J. Yoo, "Low-Power Network-On-Chip for High-Performance SoC Design," *IEEE Trans. VLSI Syst.*, vol. 14, Feb. 2006, pp. 148-160.

[22] S.M. Kim et al., "Hardware-Software Implementation of MPEG-4 Video Codec," *ETRI J.*, vol. 25, no 6, Dec. 2003, pp. 489-502.

Jaesung Lee received his BS and MS in electronic engineering from Ajou University, Suwon, Korea, in 1999 and 2001, respectively. He received his PhD in electrical engineering and computer science from Seoul National University, Seoul, Korea, in 2008. He has been with the Computer System Department of ETRI, Daejeon, Korea, where he is currently working as a senior engineer, since 2001. His research interests include VLSI architecture, network processor design, and multimedia SoC and SiP design. He has been the named author of more than 30 published papers in international and domestic journals and proceedings. He has held more than 10 overseas and domestic patents. He won the best paper award at the 8th Samsung Humantech Thesis Prize in 2002. Dr. Lee is a member of the IEEE, the IEEK, the KIISE, and the KICS.