ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.24, 1–12

Design of an application specific instruction set processor for a universal bitstream codec

Seung-Hyun Choi[1], Tae-Moon Roh[2], Yong Ho Song[3], and Seong-Won Lee[1a)]

[1] *Department of Computer Engineering, Kwangwoon University, 20, Gwangun-ro, Nowon-gu, Seoul, 139–701, Korea*

[2] *Electronics and Telecommunication Research Institute, 218 Gajeong-ro, Yuseong-gu, Daejeon, 305–700, Korea*

[3] *Department of Electronic Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul 133–791, Korea*

a) *swlee@kw.ac.kr*

Abstract: Recent increases in the type and amount of multimedia data has required a versatile device enough to play various data. Especially a variety of lossless compression algorithms often causes large amount of redundant computations. In this paper, an application-specific instruction processor tailored to effectively process such coding algorithms is proposed. The functionality and performance of the processor has been verified by using it to run the H.264/AVC baseline profile encoder. The experimental results show that the proposed processor can save about 96% and 36% of the execution cycles of ARM Cortex-A9 and Intel i7 processors, respectively.

Keywords: ASIP, entropy coding, RLC, bitstream, H.264

Classification: Integrated circuits

References

[1] ARM: ARMv7 architecture: http://www.arm.com.

[2] Intel: IA-32 architecture: http://www.intel.com.

[3] J. J. Lee, J. Y. Lee, M. K. Jeong, S. M. Park and N. W. Eum: ITC-CSCC (2008) 189.

[4] I. G. Heo, S. S. Park, J. Y. Lee and Y. H. Paek: ISOCC (2010) 186. DOI:10.1109/SOCDC.2010.5682942

[5] M. Schwalb, R. Ewerth and B. Freisleben: IEEE Trans. Multimed. **11** (2009) 1. DOI:10.1109/TMM.2008.2008873

[6] J. H. Han, M. Y. Lee, Y. Bae and H. Cho: ETRI J. **27** (2005) 491. DOI:10.4218/etrij.05.0905.0001

[7] R. Husemann, A. A. Susin, R. Kintschner, J. Valdeni and V. Roesler: MWSCAS (2011) 1. DOI:10.1109/MWSCAS.2011.6026288

[8] K. W. Kim, S. H. Park and Y. H. Paek: ISOCC (2009) 373.

[9] O. Sohm: *Variable-Length Decoding on the TMS320C6000 DSP Platform* (Application Report of Texas Instruments, 2002) 7.

[10] Y. H. Kim, Y. J. Yoo, J. H. Shin, B. H. Choi and J. K. Paik: IEEE Trans. Consum.

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

Electron. **52** (2006) 943. DOI:10.1109/TCE.2006.1706492
[11] D. Lu, G. Liu and L. Zhu: IPTC (2011) 79. DOI:10.1109/IPTC.2011.27
[12] S. D. Kim and M. H. Sunwoo: J. Signal Processing Systems **50** (2008) 53. DOI:10.1007/s11265-007-0109-y
[13] J. H. Seo, H. H. Jo, D. G. Sim, D. H. Kim and J. H. Song: EURASIP J. Image Video Process. **2013** (2013) 23. DOI:10.1186/1687-5281-2013-23
[14] Y. S. Ko, Y. M. Yi and S. H. Ha: DASIP (2011) 11. DOI:10.1109/DASIP.2011.6136860
[15] R. R. Osorio and J. D. Bruguera: IEEE International Conf. on ASAP (2007) 222. DOI:10.1109/ASAP.2007.4429984
[16] D. Moolenaar: Microprocessor forum (2007) http://ip.cadence.com/uploads/pdf/MPF07_MVP_final.pdf.
[17] M. Kimura, K. Iwata, S. Mochizuki, H. Ueda, M. Ehama and H. Watanabe: IEEE Micro **29** [6] (2009) 18. DOI:10.1109/MM.2009.87
[18] CEVA: CEVA-MM3000 product brief (2011) http://www.ceva-dsp.com.
[19] C. A. Pinto, A. Beric, S. P. Singh and S. Farfade: IEEE ISM'06 (2006) 493. DOI:10.1109/ISM.2006.83
[20] A. L. F. M. Drijvers: MS thesis Eindhoven University of Technology, Eindhoven (2008).

1 Introduction

Multimedia devices such as smartphones, smart pads, and digital cameras are widely used in daily life. These devices are often versatile enough to play many different types of video contents, each conforming to one of video standards such as MPEG-1/2/4, H.264/AVC, Divx, VP4 or WMV. Due to such diversity in video format, multimedia devices are required to support multiple codecs.

One of the key functions of video codecs is to compress and decompress video data for efficient transmission and storage. To this end, the codecs use both lossy and lossless compression techniques: the lossy compression techniques remove time and space redundancy in multimedia data, while the lossless compression techniques reduce the content size by eliminating the statistical redundancy in the data. Whenever a new standard is introduced, the data compression rate, image resolution and quality are significantly improved over previous ones. However, the improvement often increases the computation requirement of codec algorithms even though the core compression algorithms are almost the same.

Entropy coding (EC) and run-level coding (RLC) are popular lossless compression techniques used in video compression standards. The EC compresses video data by converting fixed-size symbols into variable-size ones in such a way that it assigns short output symbols to frequently-used source symbols. On the other hand, the RLC compresses video data by converting pixel-rate data into symbol-rate data: it presents a sequence of identical symbols using a pair of the symbol and its occurrence count. The RLC is effective in reducing data size where the same symbols consecutively appear in video data.

This paper proposes a novel ASIP architecture tailored to effectively execute the EC and RLC operations. The proposed architecture includes new application-specific instructions and the internal microarchitecture to support the efficient

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

execution of the instructions. This architecture can be used to implement various video standards.

The functionality of the proposed architecture has been verified by implementing the processor in a VLSI chip with an 80-nm process. The performance of the H.264/AVC codec running on the proposed processor was measured with 3 QCIF video streams in baseline profile. The experimental results show that the proposed architecture saves the execution cycle up to 96% and 36%, when compared to ARM Cortex-A9 [1] and Intel i7 [2] processors, respectively. It was also observed that the VLSI implementation requires only a small number of silicon gates.

2 Background and related work

2.1 Entropy coding and run-level coding

As aforementioned, the EC and RLC are the lossless coding schemes used in various video standards. The EC compresses video data using the non-uniform statistical probability of symbol occurrence. Some famous EC algorithms are Huffman Coding, Arithmetic Coding, and Exp-Golomb Coding. The Huffman Coding and Arithmetic Coding are used in CAVLC and CABAC of the H.264/AVC, respectively. The Exp-Golomb is used to compress parameters other than video data.

The core operations of RLC are zigzag scan and run-level pairing. The scan order is chosen depending on which standard is in use and which scan order is effective for increasing the number of consecutive 0s. An array of scanned values contains one or more cluster of a non-zero value followed by strings of zeros. The RLC reduces the space required to store scanned values by using two- or three-dimensional run-level pairing. In the two-dimensional pairing, an array of scanned values is presented with runs and levels, where the run is the number of consecutive 0s between non-zero coefficients, and level is the non-zero coefficient. In the three-dimensional run-level pairing, symbols are presented with three-element tuples: run, level and last. MPEG1 and MPEG2 use the two-dimensional pairing, while MPEG4 and H.264 employ the three-dimensional one.

As an attempt to enhance the performance, it is possible to pipeline EC and RLC operations. In this case, the EC produces a series of symbols into a buffer which is then used as an input source for the RLC. However, due to the difference in processing rate between EC and RLC, the pipeline mechanism should be carefully designed to incorporate a sufficient amount of buffer.

2.2 Acceleration techniques

Many recent researches have proposed application-specific instruction processor (ASIP) architectures to efficiently run software video codecs. The proposed architectures provide special instructions suitable for the execution of codec-specific operations [3]. They have focused on the reduction of memory accesses to speed up the execution of motion compensation, motion estimation [4, 5], and intra- and inter-prediction [6, 7, 8, 9]. One of the other approaches is to provide an efficient table lookup operation for the EC and RLC [10, 11].

In addition, an ASIP approach for a H.264/AVC decoder uses a dedicated hardware accelerator for entropy decoding [12]. Another ASIP used for imple-

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

mentation of H.264 decoder [13] utilizes some special bit-patterns found in the entropy decoding tables of H.264 to accelerate its processing speed. However, the use of such special properties hinders the processor from being used for entropy encoding.

There are also some approaches that accelerate codec execution by using multi-core CPUs or both CPUs and GPUs [14]. They effectively run most codec functions, but not the EC which is known to be hard to parallelize.

The complexity of various entropy encoding and decoding algorithms can be handled by pipelined array processors [15]. Each processor of this architecture provides only simple instructions that can be used to execute the EC efficiently. But the cost of array processors and communication infrastructure is considerably high compared to a dedicated hardware accelerator.

3 Proposed architecture

The proposed ASIP, called Bit Stream Processor (BSP), consists of two modules, Syntax Processor (STP) and Run-Level Engine (RLE), each of which responsible for syntax parsing and EC, and RLC, respectively. This processor excludes the signal processing unit for the lossy coding part of the video compression. It is because the signal processing function could be run in software on an external microprocessor. In addition, the BSP works with an external DMA which is in charge of transferring encoded bitstream data and decoded video data to the next processing unit.

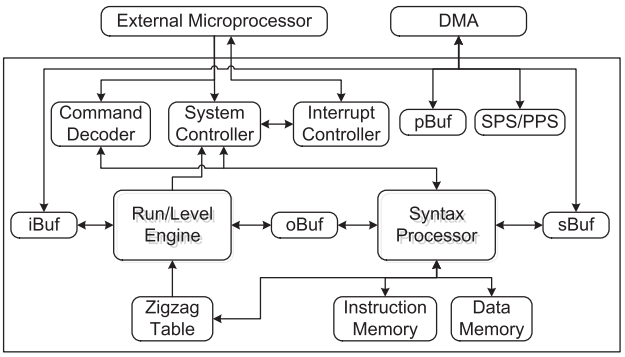


Fig. 1. Proposed BSP architecture

The overall architecture of the proposed processor is illustrated in Fig. 1. The STP and RLE are connected with the other components via four buffers (iBuf, oBuf, sBuf, and pBuf) to store intermediate data. The iBuf delivers DCT coefficients between the external microprocessor and RLE. The oBuf stores run/level pairs between RLE and STP. The sBuf holds the bitstream between STP and DMA. The pBuf, located between the external microprocessor and STP, transfers codec metadata (parameters) that do not need to pass through RLE. The RLE and STP can run in an asynchronous way. However, the STP can control the operations of RLE by setting the proper command registers of RLE. It communicates with the system controller through the special status registers (which is not shown in the figure). The use of buffers allows STP and RLE to run in a pipelined way.

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

3.1 STP for entropy coding

The STP is a 32-bit RISC processor with Harvard architecture. It contains special registers and a barrel shifter that are customized for the entropy coding operations. It has two internal pipeline stages: instruction fetch stage and execution stage.

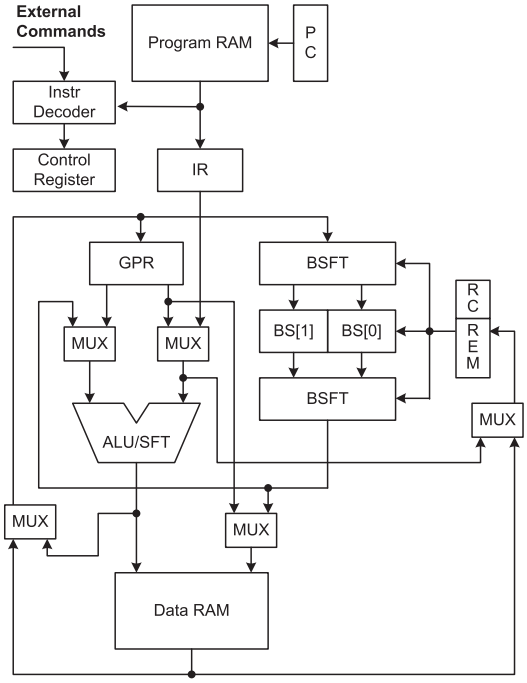


Fig. 2. STP data path

The data path inside the STP is shown in Fig. 2. The barrel shifter (BSFT) is responsible for shifting an incoming value with a specified amount and the bitstream registers (BS[0],[1]) are used to store a portion of bitstream data under processing. The remainder register (REM) and its status register (RC) stores the calculated size of the processed symbol. The other components in the STP are a program counter (PC), an instruction register (IR), general purpose registers (GPR), and functional units for arithmetic and shift (ALU/SFT).

The new instructions designed for entropy coding are listed in Table I. The TLE (table lookup for encoding) and TLD (table lookup for decoding) instructions access a lookup table during the encoding and decoding operations, respectively. The TLD instruction locates a symbol in the entropy coding table for the current symbol in the bitstream and converts it to the corresponding DCT coefficients. It then performs shift and match operations to search for the next symbol.

When a symbol is used as an index to lookup the table, the lookup table needs to be large enough to have 2^n entries, where n is the maximum width of a symbol. However, it is observed that symbols usually have many leading 0s, which means that most of the table entries are never accessed. In order to reduce the table size, it is possible to define small lookup tables each holding a sequence of entries that are actually accessed. In this case, a sequence of entries in the same lookup table is called *class*, and the leading bits of a symbol is used as an identifier to the class that the symbol belongs to. In the proposed architecture, the index is further shortened

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

in length by grouping the leading 12 bits of a symbol into three nibbles and collapsing each nibble into a bit. The collapse operation is done by ORing all the bits in each nibble to form a 3-bit class number.

Table I. Bitstream operation

Instruction	Operation
TLD dr,sr,(imm1),imm2	index <= sr + {or(BS[0:3]), or(BS [4:7], or(BS[8:11], BS[31 - imm2 : 31], (imm1)} dr <= code(mem[index]) RC/REM <= REM+ length(mem[index])
TLE sr1, sr2	index <= sr1 + sr2 BS <= code(mem[index]) RC/REM <= REM+length(mem[index])
LZS dr,sr	dr <= leading0s(BS) RC/REM <= REM+leading0s(BS)
LOS dr, sr	dr <= leading1s(BS) RC/REM <= REM+leading0s(BS)
REM sr	RC/REM⟨5:0⟩ <= REM⟨5:0⟩+sr⟨5:0⟩
LBS sr, imm	BS <= mem[sr]; sr <= sr+imm
LBC sr, imm	if (RC), BS <= mem[sr] sr <= sr+imm // auto-indexing
STS sr, imm	mem[sr] <= BS; sr <= sr+imm

For example, if the leading 12 bits are all 0, then the class number becomes 000. If they are 0000_0001_0001, the class number is 011. Then, the class 000 uses the remaining bits of the symbol as an index. Likewise, the class 001 uses the least four bits of the leading 12 bits as an index in conjunction with the remaining bits, and so on.

The class number is concatenated as a prefix to the remaining bits of the symbol to form an index for the lookup tables. The imm operand of TLD instructions is divided into imm1 and imm2: imm1 is a table prefix for few special symbols with 14 or more leading zeros and can be omitted in most cases, while imm2 is a mode number between 1 and 8 to determine how many bits from the BS are used for table indexing. Since the maximum symbol length in each table is known, using imm2 reduces the table size.

Other instructions, such as REM (add to remainder register), LBS (load to BS register), LBC (conditional load to BS register), STS (store BS register), STC (conditional store BS register), LZS (leading zeros in BS register) and LOS (leading ones in BS register), are used to manipulate bitstreams, whose semantics are summarized in Table I. In the Table I, or() is ORing all bits, code() the code part of the symbol table, and length() the length part of the symbol table.

The proposed STP performs better for entropy coding than a general purpose CPU, since it provides both general RISC instructions and special instructions for bitstream processing. The use of STP allows assembly codes to be written in order to process bitstreams of various standards. Its flexibility will be useful in supporting future standards.

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

3.2 RLE for RLC and zigzag scan

The proposed RLE is a hardwired module which is responsible for performing zigzag scan and run level coding. There are several tables in the module that can be used in zigzag scanning (ZigZag TBL). In addition, the module has the other components: a counter to hold an address to the ZigZag TBL (Addr CNT), another counter to count the number of continuous 0s (Zero Counter), iBuf and oBuf to temporarily store encoded and decoded coefficients, respectively, and a controller to orchestrate the operations of each component.

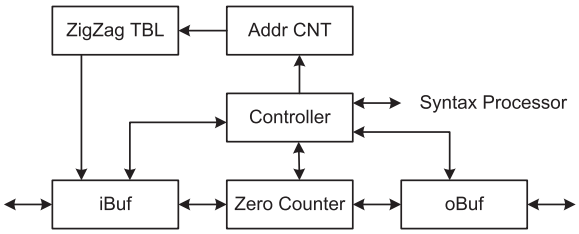


Fig. 3. Basic block diagram of RLE

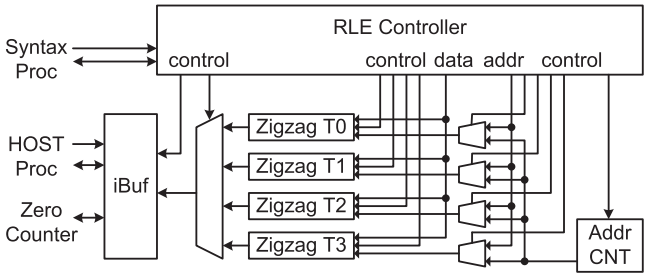


Fig. 4. Zigzag table control

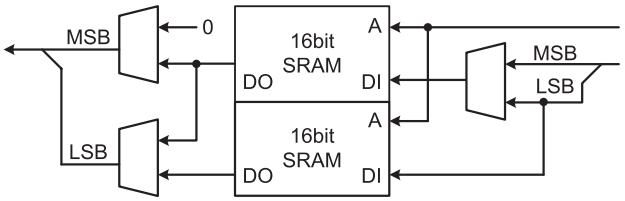


Fig. 5. oBuf configuration

Fig. 3 shows the entire block diagram of the RLE. The contents of the ZigZag TBL, which is the zigzag scan order, is written by the STP. The raster data in iBuf are then read or written in the zigzag scan order. Since the RLE has a hardwired finite state machine, the scan order change quite limits the flexibility. In order to handle all the zigzag scan patterns of various video standards, four programmable tables are used to store various zigzag scan orders. The RLE identifies the codec type from the program field of its control register and determines the zigzag scan patterns based on the given codec type. As shown in Fig. 4, the zigzag order and the corresponding macroblock addresses are available from the four ZigZag TBLs, and all the table accesses including programming tables and selecting a table to use are controlled by the RLE Controller according to the STP's commands.

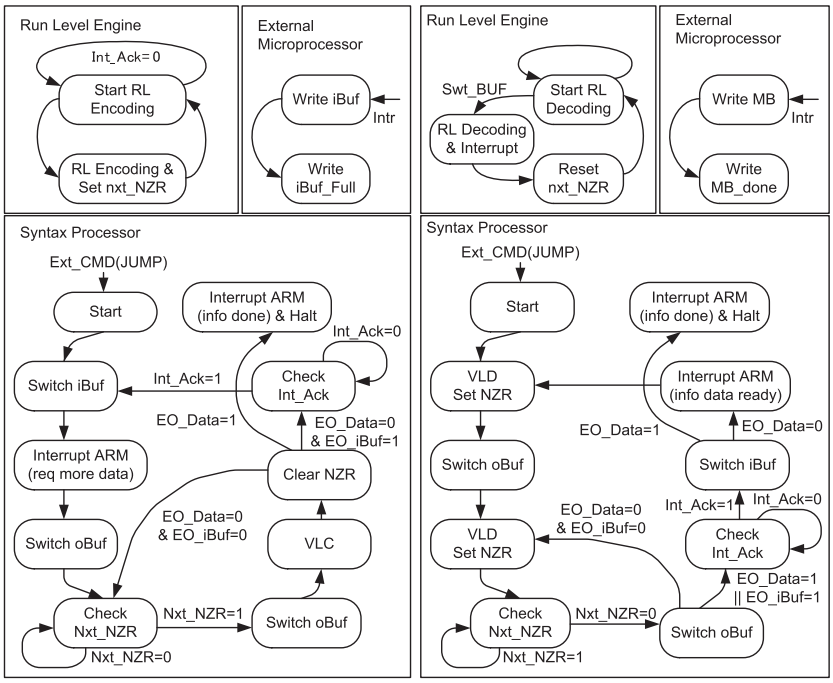
DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

As shown in Fig. 5, oBuf consists of two 16-bit SRAM memories. This buffer can be used to implement EC for both H.264 and MPEG1/2. In H.264, one of two SRAMs is accessed to read Run and Level, while in MPEG2, both SRAMs are simultaneously accessed to read Run and Level. The difference in buffer access comes from the different ordering of Run and Level in memory. In the RLE design, the RLE is programmable in such a way that the data order stored in oBuf can be configured by the STP.

Each of iBuf and oBuf is a dual buffer, and therefore it can receive RLC data alternatively during the RLE operation. It means that the STP can continue to operate regardless of the status of the RLE.

The time required for the RLE to encode a single macroblock includes the cycles for initialization, the number of pixels in the macroblock, and the cycle for buffer change. All the data are transferred from iBuf to oBuf during the encoding phase and from oBuf to iBuf during the decoding phase.

3.3 RLE synchronization with STP



(a) (b)

Fig. 6. Synchronization state diagram. (a) Encoding. (b) Decoding.

The System Controller informs the STP of data availability by setting corresponding bits of the system command register of the STP. However, the STP and RLE run asynchronously, and therefore the STP needs to know whether the RLE operation is completed by checking the status of the completion flag of the NZR register in oBuf. When the current buffer of oBuf becomes full, the RLE sets the flag to 1 and switches to the next buffer. When both buffers are full, it stops any further operations and sends a busy signal to the external microprocessor.

Fig. 6(a) illustrates the state diagram of the synchronization flow between STP, RLE and System Controller. The STP checks the dual buffer register before starting

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

its operation, particularly the nxt_NZR flag that indicates the status of the other buffer that is not being accessed. It also checks the MB_done flag to see if the System Controller finishes writing macroblock (MB) data to the buffer. If the flag is set, the STP switches to the other buffer. Next, the RLE starts encoding a new RLC while the System Controller brings a new MB data to the dual buffer on the opposite side. Fig. 6(b) depicts a state diagram of the synchronization procedure among the STP, RLE and System Controller for the decoding operation.

The STP checks the flag of ~nxt_NZR and MB_done, and then starts the operation. That is, when the RLC finishes a decoding procedure and when the System Controller finishes reading an MB, the STP switches to the next buffer. The RLE then starts to decode a new RLC, while the System Controller takes a new MB from the other buffer.

4 Experimental results

The proposed architecture was implemented using both FPGA and VLSI to measure real-time performance. A mobile processor was used as the System Controller, and the baseline profile of the H.264 codec was assembly-programmed for execution on the STP. All the buffers except sBuf are implemented to handle 64 macroblocks in a single turn. The sBuf is made of 32-bit 1024 words.

For the performance evaluation, three QCIF images: Foreman, Spider, and Violet were used. Fig. 7 shows the test images. The data file for Spider is 2–3 times larger than those for Foreman and Violet due to fast motion changes.

(a) (b) (c)

Fig. 7. Benchmark images used for evaluation: (a) Foreman, (b) Spider, (c) Violet

It is difficult to directly compare the performance of the proposed processor and those of other microprocessors, because they have different instruction sets. Therefore, the number of execution cycles required to encode or to decode the benchmark images were measured on the proposed processor, on an Intel processor, and on an ARM processor.

For testing Intel IA32 instruction set, an Intel i7 3770 processor with 1.6 GHz is used in the experiment. It has quad cores with Hypertheading, and run 8 threads concurrently. For ARMv7 instruction set, an ARM Cortex-a9 processor with 866 MHz that runs a single thread is used. The performance of the proposed processor is obtained by analyzing the simulation results for the VLSI implementation with the system clock of 200 MHz.

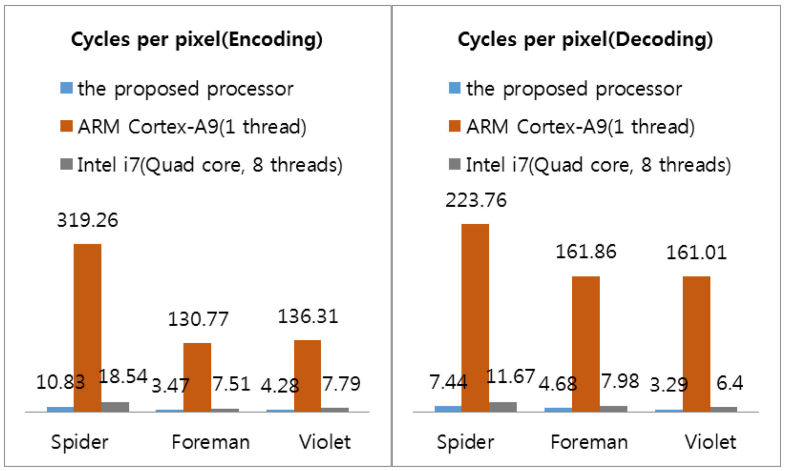
DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

The cycles per pixel is calculated with the average execution time out of 10 experiments which is to measure the execution time of the entropy coding/decoding routines in the JM 18.6 reference software. We use the gcc 4.8 compiler with O3 optimization option as instructed in the JM software. The performance of the Intel and ARM processors was measured with the JM 18.6 reference software.

Table II and Fig. 8 summarizes the result of the image testing. Fig. 8 shows the comparison of times to generate bitstream and to analysis bitstream in JM software on the Intel processor, ARM processor, and the proposed processor.

Table II. Test image information in QCIF

Test Sequence	Number of frames	Average number of symbols per frame
Spider	250	6690.9
Foreman	300	2602.8
Violet	100	2293.2



(a) (b)

Fig. 8. Cycles per pixel: (a) Encoding, (b) Decoding

Table II shows the total number of cycles per pixel in each processor, and Fig. 8 compares the number of cycles measured in the three processors. As shown in the figure, the proposed processor uses 3.29 to 10.83 cycles per pixel, while an Intel i7 processor needs 6.4 to 18.54 cycles and an ARM Cortex-A9 processor needs 130.77 to 319.26 cycles. As previously mentioned, it is difficult to make a side-by-side comparison between processors with different instruction sets. However, considering that the Intel i7 processor used in the experiments can issue up to eight instructions per cycle, the actual accumulated cycles, when added together, could be up to eight times longer. The ARM Cortex-A9 processor requires around 29 times more cycles than the proposed architecture. The Spider image requires three times longer codes for entropy coding than the other images. The STP successfully processed entropy coding, producing a small difference in cycles between Spider and the other images.

The proposed processor is also compared to the existing video processors that support multiple standards. Since most video processors use VLIW (Very Long

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

Instruction Word) architecture for stream processing, it is difficult for direct comparison with the proposed processor that has a single pipe. However, the proposed processor has a RISC instruction set and it is easy to adopt VLIW architecture. By considering the issue width of the existing video processors, the performance of the proposed processor can be compared with them roughly.

Table III. Cycles per pixel (Decoding)

Name	Frequency	Max Image Size	Issue width	Max Cycles per pixel
Tensilica Xtensa [16]	162 MHz	720 × 480 30p	3	15.63 × 3 = 46.89
Kimura's Processor [17]	162 MHz	1920 × 1080 30p	2	2.6 × 2 = 5.2
CEVA-MM3000 [18]	upto 250 MHz	1920 × 1080 30p	4	4.02 × 4 = 16.08
Silicon Hive HiveFlex [19, 20]	200 MHz	1920 × 1080 30p	3	3.21 × 3 = 9.63
Proposed processor	196 MHz	1920 × 1080 30p	1	3.29 ∼ 7.44

Table III shows the cycles per pixel for the existing video processors to process video bitstream. While considering the number of concurrent issue, Tensilica Xtensa [16], CEVA-MM3000 [18], and Silicon Hive HiveFlex [19, 20] need more cycles per pixel as compared to that of the proposed processor. Kimura's processor [17] shows comparable performance to the proposed processor in the expense of an out-of-order execution hardware. It should be considered that adding an out-of-order execution hardware to a VLIW processor is quite expensive. The comparison shows that the additional instructions for stream processing are effective to improve the performance of entropy coding.

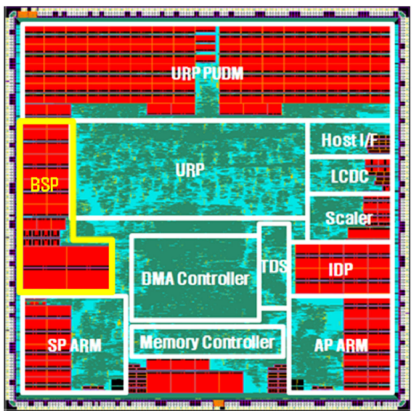


Fig. 9. BSP layout in U-chip

From this result, it is expected that the proposed architecture can handle high-bitrate videos such as HDTV and UHDTV by efficiently processing a significant amount of bitstream data.

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014

The bitstream processor proposed in this paper was implemented in VLSI as a part of video codec SoC, called U-chip. Fig. 9 shows the SoC layout. The bitstream processor is annotated as BSP on the layout. The U-chip was manufactured by an 80-nm process. The BSP occupies 3.818 mm^2, of which the memory occupies 3.725 mm^2. The proposed processor has two pipeline stages and is designed to run up to 200 MHz, and the BSP chip implemented runs with 192 MHz clock. Since a large buffer was needed to enable simultaneous encoding and decoding of 64 macroblocks, a large memory area was allocated in the implementation. The whole processor uses approximately 1.353 million gates, of which the number of gates of BSP logic, except the memory, is approximately 33 kilo gates. If the DMA and interrupt controller are used only for the BSP, it is possible to reduce memory size to 59% of the current implementation. In this case, the total number of gates would be 0.813 million gates, and the area would be as small as 2.23 mm^2.

5 Conclusion

In this paper, a new bitstream processor, called BSP, is presented based on ASIP technology. The BSP consists of two internal components: STP and RLE. The STP provides instructions for efficient entropy encoding and decoding, and the RLE efficiently performs zigzag scan and RLC. The STP reduces the size of the lookup table for variable-length codes. The use of both STP and RLE is capable of decreasing about 96% and 36% of the execution cycles, when compared to ARM Cortex-A9 and Intel i7 processors, respectively.

The proposed architecture allows STP and RLE to run in parallel in a pipelined fashion, and therefore bridges the gap in processing rate between pixel rate processing and symbol rate processing. The STP and RLE are connected using a dual buffer mechanism. The proposed processor was implemented into a video codec SoC and tested with the most complicated H.264 encoding/decoding algorithms.

It was also observed that the EC and RLC, which are commonly used in various standards, were efficiently performed in the proposed hardware architecture. In future research, repetitive patterns in data processing will be analyzed to add more instructions for efficient processing.

Acknowledgments

This work was partly supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2014-H0301-13-1018), by the Industrial Strategic Technology Development Program (10049095, Development of Fusion Management Platform and Solutions for Smart Connected Devices) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) and by Research Grant of Kwangwoon University in 2013.

DOI: 10.1587/elex.11.20141047
Received November 6, 2014
Accepted November 11, 2014
Publicized November 28, 2014
Copyedited December 25, 2014