Scalable Design Framework for JPEG2000 System Architecture

Hiroshi Tsutsui[1], Takahiko Masuzaki[1], Yoshiteru Hayashi[1], Yoshitaka Taki[1], Tomonori Izumi[1], Takao Onoye[2], and Yukihiro Nakamura[1]

[1] Department of Communications and Computer Engineering, Kyoto University
Yoshida-hon-machi, Sakyo, Kyoto, 606-8501 Japan
{tsutsui, masuz, teru, taki}@easter.kuee.kyoto-u.ac.jp,
{izumi, nakamura}@kuee.kyoto-u.ac.jp
[2] Department of Information Systems Engineering, Osaka University
2-1 Yamada-Oka, Suita, Osaka, 565-0871 Japan
onoye@ist.osaka-u.ac.jp

Abstract. For the exploration of system architecture dedicated to JPEG2000 coding, decoding and codec, a novel design framework is constructed. In order to utilize the scalability of JPEG2000 algorithm aggressively in system implementation, three types of modules are prepared for JPEG2000 coding/decoding/codec procedures, i.e. software, software accelerated with user-defined instructions, and dedicated hardware. Specifically, dedicated hardware modules for forward and inverse discrete wavelet transformation (shortly DWT), entropy coder, entropy decoder, and entropy codec as well as software acceleration of DWT procedure are devised to be used in the framework. Furthermore, a JPEG2000 encoder LSI, which consists of a configurable processor Xtensa, the DWT module, and the entropy coder, is fabricated to exemplify the system implementation designed through the use of proposed framework.

1 Introduction

The increasing use of multimedia information requires image coding system to compress different types of still images with different characteristics by a single processing flow besides attaining high coding efficiency. To fulfill this requirement, JPEG2000 is currently being developed by ISO/ IEC JTC1/SC29 WG1 (commonly known as the JPEG), and JPEG2000 Part I[1] was standardized in January, 2001. Distinctively, in JPEG2000, discrete wavelet transformation (shortly DWT) is adopted to decorrelate images spatially to improve compression efficiency. With the use of this transformation, so-called *embedded stream* can be generated, in which code for low quality/bitrate image is included in that for high quality/bitrate image. Therefore, JPEG2000 can be regarded as the viable image coding scheme in the coming network era to be used in a variety of terminals for different application fields. However, this fact also implies that performance requirement for terminals and applications varies widely. Thus any of single software or hardware implementation can hardly fulfill performance

P.-C. Yew and J. Xue (Eds.): ACSAC 2004, LNCS 3189, pp. 296–308, 2004.

requirements for all range of terminals and applications. On the other hand, it is also impossible to arrange a full set of customized implementations for all of terminals or applications in terms of man-power resources.

Motivated by this tendency, we propose a novel framework of JPEG2000 system architecture, providing the distinctive ability of architectural exploration in accordance with the specification of each terminal. Through the use of this framework, an efficient JPEG2000 system organization is obtained by referring to performance requirements and limitations for implementation. The proposed framework is based on Tensilica's configurable processor Xtensa[2], which has an ability to be customized for a specific application by equipping user-defined instructions described in Tensilica Instruction Extension (TIE) language[3]. Enhancing this distinctive feature to such an extent to equip specific hardware modules prepared for procedures in JPEG2000, our framework provides scalable solution for JPEG2000 system architecture.

For each procedure of JPEG2000 coding and/or decoding, either of software implementation, software implementation accelerated by user-defined instructions, or hardware implementation is selectively employed. The implementation scheme for all procedures are decided by referring to performance requirements and/or limitations to design. In case extremely high processing performance is needed far more than that of hardware implementation, it is possible to equip two or more modules at the same time.

2 JPEG2000 Processing Flow

Fig. 1 depicts the procedures of JPEG2000 encoding scheme. First, a target image is divided into square regions, called *tiles*. Tiles of each color component are called *tile components*. Then 2-D forward DWT decomposes a tile component into LL, HL, LH, and HH subbands by applying 1-D forward DWT to a tile component vertically and horizontally. The low resolution version of the original tile component, i.e. LL subband, is to be decomposed into four subbands recursively. A subband is divided into *code-blocks*, each of which is coded individually by entropy coder. The entropy coder adopted in JPEG2000 is context-based adaptive binary arithmetic coder which consists of coefficient bit modeling process to generate contexts and arithmetic coding process, known as MQ-coder, to compress a binary sequence based on the context of each bit.

Decoding scheme of JPEG2000 is the reverse process of the encoding. During this process, 2-D backward DWT is realized by applying a series of 1-D backward DWTs (horizontal 1-D DWT and vertical 1-D DWT) to a tile component also in the reverse order of 2-D forward DWT. The set of filter coefficients used in 1-D backward DWT is the same as that in the forward DWT. Fig. 2 depicts the entropy coding and decoding procedure. Coefficient bit modeling is a common process to encoding and decoding. MQ-decoder extracts binary sequences from compressed data referring to contexts generated by the coefficient bit modeling process, while MQ-coder generates compressed data from contexts and binary sequences.

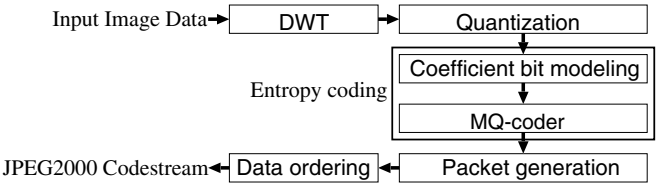

Fig. 1. Block diagram of JPEG2000 encoding

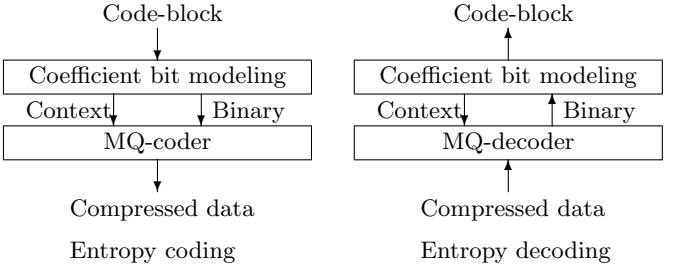

Fig. 2. Entropy coding and decoding

3 JPEG2000 System Framework

Our proposed framework is distinctive in that an efficient JPEG2000 encoding and decoding system architecture can be explored by selecting implementation scheme for each procedure considering performance requirements such as image resolution or processing throughput and limitations to design such as power consumption, process technology rule, or chip size. Three types of implementation schemes are prepared in our framework; software implementation, software implementation accelerated by user-defined instructions added to Xtensa, and dedicated hardware implementation. As for hardware implementation, multiple modules can be used at the same time if necessary. To embody such a Plug-and-Play like feature, each hardware module is designed to have a generic SRAM-based interface which can support various bus architectures by only designing interface converters. Therefore, our framework makes it much easier to design a JPEG2000 encoding and decoding system than conventional tedious manual design tasks of each procedure, which would be implemented as software or hardware. For forward/backward DWT and entropy coding/decoding, common hardware components are prepared. These procedures handle relatively large square regions of an image, called tile and code-block, respectively. Such a procedure is inherently suitable as a component, and the overhead of data transfer between the hardware component and memory can be concealed when direct memory access (DMA) is applied effectively. In addition to these common hardware components, an software module accelerated by user-defined instructions is prepared for DWT. Since the dominant operation in DWT procedure is filter operation, a set of instructions for filter operation is added to the Xtensa's

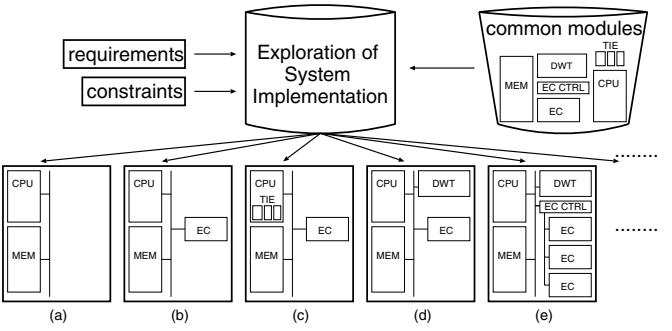

Fig. 3. Basic structure of the proposed framework

original instruction set so that considerable performance improvement can be achieved without any additional hardware component.

Moreover, to organize JPEG2000 codecs, an entropy codec hardware component is prepared. This component is smaller than the simple combination of the entropy coder and decoder in terms of the number of gates. As for DWT, the differences between encoding and decoding are ordering of two 1-D DWTs applied in series, ordering of filter coefficients, and signs of several filter coefficients. The first one seems to affect largely the DWT architecture. However, in the case of lossy compression, the error introduced by employing same DWT ordering for both forward and backward DWTs is quite slight. Therefore we employ the same ordering of DWTs to make it easy to design forward and/or backward 2-D DWT. As for other differences, we simply introduce multiplexers to select the signs of filter coefficients, etc.

Fig. 3 illustrates the basic structure of the proposed framework. Each system implementation exemplified in Fig. 3 is briefed below. Though only the case for JPEG2000 encoders is explained, JPEG2000 decoders and codecs can be implemented by the same way of the encoders.

Fig. 3(a) is a system implementation example where all modules are implemented as software so that the system is composed only of a CPU and a main memory. This solution is used when it is impossible to employ hardware modules to the system due to die size limitation, the processing speed is not the key point aimed at, or the CPU provides sufficient performance for application.

Fig. 3(b) is a system implementation example where only the entropy coding (shortly EC in the figure) procedure is implemented by a hardware module. The hardware implementation of the entropy coding makes a large contribution to improvement of the speed, since this procedure is the most computationally intensive one among all the procedures as its detailed discussion is given in the next section.

Fig. 3(c) is a system implementation example where the entropy coder is implemented by hardware module, and instructions for DWT's filter operations are added to Xtensa. Since DWT procedure handles a whole tile, the number of cycles needed to execute the procedure in DWT, except for filter operations, such

as address calculation, memory accessing, and so forth, is still large. Performance improvement in comparison with Fig. 3(b), however, can be achieved without any additional hardware module.

Fig. 3(d) is a system implementation example where both the entropy coder and DWT are implemented as hardware modules. In addition to the performance improvement as the benefit of these modules, memory accessing bandwidth is reduced.

Fig. 3(e) is a system implementation example where DWT is implemented by hardware module, and the entropy coding procedure is implemented by multiple hardware modules. This solution is the fastest among Fig. 3(a) through (e), and attains very high throughput. In this case, an additional controller is needed to manage these modules.

In this manner, our framework successfully provides scalable solution for JPEG2000 system architecture with the use of common modules.

4 Analysis of JPEG2000 Encoding

To construct the framework, first we implemented software JPEG2000 encoder and decoder, and analyzed computational costs of all procedures. Since DWT is to be executed in fixed point arithmetic in our software, even embedded CPUs which has no floating-point unit (FPU) can execute this software without any additional cycles needed for floating arithmetic emulation. The main specifications of our software are summarized in Table 1, and the result of profiling encoding process by the instruction set simulator (shortly ISS) of the target CPU Xtensa using a test image LENA is summarized in Table 2. The function `encode` is to encode whole image and does not include the routines for input/output from/to a file. In the function `entropy_coding`, the entropy coding including coefficient bit modeling and arithmetic coding by MQ-coder is executed. The function `FDWT_97`

Table 1. Main specification of the JPEG2000 software

tile size	128×128
DWT	9/7 irreversible filter
DWT decomposition level	3
code-block size	32×32

Table 2. Result of profiling encoding process

%	function name	self cycle	child cycle	total cycle	call
99.5	`main`	0.13	1371191.89	1371192.02	1
96.3	`encode`	540.26	1326074.65	1326614.91	1
64.4	`entropy_coding`	84.50	887680.17	887764.67	304
20.2	`FDWT_97`	29298.20	249056.71	278354.91	16
14.0	`FILTD_97`	191668.46	771.41	192439.87	7168

unit of number of cycles is Kcycle

is to execute DWT on a tile component and includes `FILTD_97` for 1-D DWT on an array.

According to this table, it can be said that the coefficient bit modeling and arithmetic coding procedures occupy 64.4% of total encoding cycles, DWT procedure occupies 20.2% of total encoding cycles, and `FILTD_97` function occupies 69.1% of DWT processing cycles.

5 JPEG2000 Processing Modules

5.1 DWT Module

DWT hardware. When implementing DWT as dedicated hardware, the essential factor to be considered is memory organization for storing intermediate data during recursive filter processing. There are two methods to store intermediate data for DWT. One is to store the data to the main memory or a tile buffer which is placed in a DWT hardware module. In this case, whenever vertical and horizontal DWT is attempted, the data is read from the memory/buffer and the transformed coefficients are written back to it. The other is the so-called line-based method[4] which is to store the data to a line buffer containing several lines in a tile. In this case, vertical and horizontal DWT can be done at the same time by utilizing the line buffer, so that this method requests less amount of data transfer over a bus than that using the main memory. Thus, we adopted line-based method to implement DWT hardware module.

To implement DWT filters, we adopted straightforward finite impulse response (FIR) filter, where, for calculating each transformed coefficient, weighted addition of a coefficient sequence with the filter length is executed. Assuming that the depth of input image is 8-bit, this module can implement one level 2-D DWT over a tile whose width is 128 or less. The tile size of 128×128 is reasonable since it is adopted in JPEG2000 Profile 0[5], which is intended mainly for hardware implementation.

The architecture of our DWT hardware module is shown in Fig. 4, which consists of a core module, a line buffer, and an IO controller. This core module comprises the following sub-modules; an extension module which extends a sequence of coefficients at the edges by the so-called periodic symmetric extension procedure, low-pass FIR filters whose filter length is 9, high-pass FIR filters whose filter length is 7, and a pair of shift registers which receive output data from vertical DWT and store vertically low-passed and high-passed 9 coefficients to be fed to horizontal DWT.

The line buffer consists of 13 line memories each of which is a 15-bit 128-word memory. The number of the additional 7 bits for the input bit depth, 8 bits, is large enough to realize as high precision as floating point operations. Totally 9 coefficients, all of which belong to the identical column of an image, are loaded to the core module from 9 lines of the line buffer every clock. Other 2 lines of the line buffer are used for receiving transformed coefficients from the core modules every clock. The other 2 lines of the line buffer are used for IO access between the DWT and CPU.

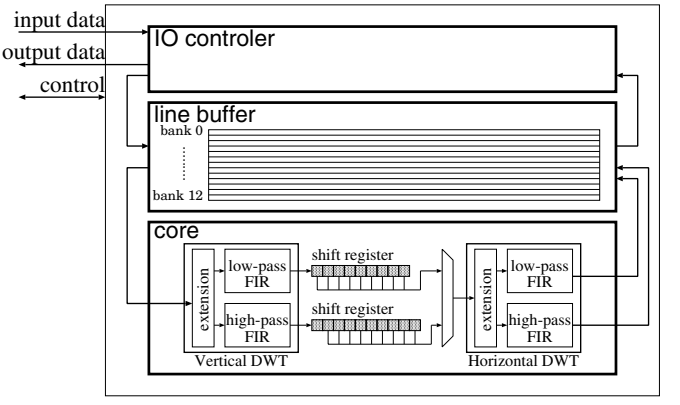

Fig. 4. Architecture of our DWT module

This core module works as follows. The 9 coefficients from the line buffer are transformed by the vertical DWT module and the results, i.e. low-passed and high-passed coefficients, are stored into the shift registers. At the same time, horizontal DWTs of vertically high-passed and low-passed sequences stored in shift registers are executed alternately, so that LL, HL, HH, and HH coefficients are output from the core module every 2 cycles.

As mentioned before, we employ the same 1-D DWT ordering for both forward and backward 2-D DWT, so as to make the architecture of forward and backward DWT almost identical. The proposed architecture of forward DWT module is designed through the use of Verilog-HDL. When this hardware module is used, the number of cycles needed for DWT with a test image LENA is 0.215 Mcycles, which is only 0.077 % of 278 Mcycles needed for software DWT. The DWT module is synthesized into 17,650 gates by using Synopsys's Design Compiler with 0.18 μm CMOS technology, with its critical path delay of 12 nsec.

Accelerated DWT software. According to the result of profiling, multiplications of fixed point variables and filter coefficients of Table 3; α, β, γ, δ, K, and $1/K$; need 6961.86 Kcycles, 7787.56 Kcycles, 7502.93 Kcycles, 6930.11 Kcycles, 6775.10 Kcycles, and 6971.17 Kcycles, respectively through entire image. Total of these values occupy 22.3% of execution cycles needed for function `FILTD_97`. Thus we implement these multiplications by user-defined instructions described in TIE. The circuits of the instructions consist of shifters and adders.

Custom instructions `MUL_A`, `MUL_B`, `MUL_C`, `MUL_D`, `MUL_K`, and `MUL_R` are to multiply positive input by lifting constants; α, β, γ, δ, K, and $1/K$, respectively. `SMUL_A`, `SMUL_B`, `SMUL_C` and `SMUL_D` are extended version of `MUL_A`, `MUL_B`, `MUL_C` and `MUL_D`, which can multiply negative input as well as positive input by lifting constants. At the final stage of lifting, target values must be shifted in right and rounded. These operations are also implemented as custom instructions. `SMUL_K` and `SMUL_R` are extended version of `MUL_K` and `MUL_R`. Same as `SMUL_[A-D]`, these

Table 3. Filter coefficiets

α	-1.586 134 342 059 924
β	-0.052 980 118 572 961
γ	0.882 911 075 530 934
δ	0.443 506 852 043 971
K	1.230 174 104 914 001

Table 4. The number of gates of user-defined instructions

instruction	#gate
MUL_A	551.75
MUL_B	401.00
MUL_C	510.25
MUL_D	489.50
MUL_K	570.50
MUL_R	367.00
other	254.25
total	3144.25

Table 5. The number of gates of user-defined instructions

instruction	#gate
SMUL_A	1085.25
SMUL_B	739.25
SMUL_C	947.00
SMUL_D	1009.25
SMUL_K	1698.25
SMUL_R	1039.00
other	439.75
total	6957.75

instructions can handle negative values with equipping a right shifter and a rounding circuit.

The result of simulation by ISS using a test image LENA shows that when MUL_A, MUL_B, MUL_C, MUL_D, MUL_K, and MUL_R are equipped, the number of cycles to call FILTD_97 function is reduced to 175.564 Mcycles (88.6%) from 192.440 Mcycles, and that when SMUL_A, SMUL_B, SMUL_C, SMUL_D, SMUL_K, and SMUL_R are implemented, the number of cycles to call FILTD_97 function can be reduced to 133.461 Mcycles (69.4%) from 192.440 Mcycles.

According to the synthesis results attained by the same manner as the hardware DWT module, the critical path delay of MUL_A, MUL_B, MUL_C, MUL_D, MUL_K and MUL_R is 6.2 nsec, ant that of SMUL_A, SMUL_B, SMUL_C, SMUL_D, SMUL_K and SMUL_R is 9.5 nsec, The numbers of gates of custom instructions are summarized in Table 4 and 5.

5.2 Entropy Coder, Decoder, and Codec

There are two reasons why the entropy coding of JPEG2000 incurs such a high computational cost. One is that a context of a coefficient on a bit plane depends on sign bits, significant states, and some reference states of the eight nearest-neighbor coefficients. Therefore, there are many conditional branches and many operations which crop variables into a bit. The other is that MQ-coder updates its internal state after compression of every one binary symbol. When JPEG2000 entropy coder is implemented as hardware, MQ-coder may become the performance bottle-neck of the total system since the MQ-coder requires at least 1 cycle to process 1 binary symbol. Needless to say, considering hardware utilization efficiency coefficient bit modeling must be implemented with the same throughput as the MQ-coder. In our hardware entropy coder, pixel skipping scheme[6] is used to attain almost ideal performance.

Entropy Coder. Fig. 5 depicts the block diagram of our entropy coder, which consists of a coefficient bit modeling module, an MQ-coder, an IO controller, a plane controller, an FIFO, and a set of buffer memories. As buffer memories, the entropy coder has a code-block buffer to store code-block data, plane buffers to store bit plane as well as reference data needed to generate contexts, and stream buffer to store encoded data called *stream*. The FIFO is used to suppress the difference of the throughputs of the modeling module and the MQ-coder.

The above mentioned entropy coder is designed by Verilog-HDL. The result of logic simulation using sample image LENA indicates that when this hardware entropy coder is employed, the number of cycles needed for entropy coding of whole image is about 3.20 Mcycles, i.e. only 0.360 % of 887.76 Mcycles needed for software entropy coding.

The critical path delay is 7.0 nsec which is concluded by synthesis the entropy coder with 0.18 μm CMOS technology. The gate counts for this module is 7,901.

Next, let us discuss the size of memory for the entropy coder. The coder requests 12 bit $\times$ (32×32) = 12,288 bit as the code-block buffer, 4 $\times$ (32×32) = 4,096 bit as 4 plane buffers, and 2 $\times$ (32×32) = 2,048 bit as the double-buffered bit plane buffer. Here, the code-block buffer must have 12 bit depth in the case that the bit depth of input image is 8, and the number of guard bits is 2, which is enough to avoid the overflow of the result of DWT. As for the stream buffer for output, 8,192 bit is large enough when quantizer's step size equals 1, according to the software simulation. Consequently, 26,624 bit of memory elements are needed in total. It must be noticed that the size of stream buffer can be reduced when implementing it as an FIFO.

Entropy Decoder. Fig. 6 depicts the block diagram of our entropy decoder, which is similarly organized as the entropy coder. The differences are that the decoder does not equip FIFO, whereas controllers, MQ-decoder, and coefficient bit modeling module of the decoder are customized for decoding. Our MQ-decoder returns a binary to coefficient bit modeling module in 1cycle.

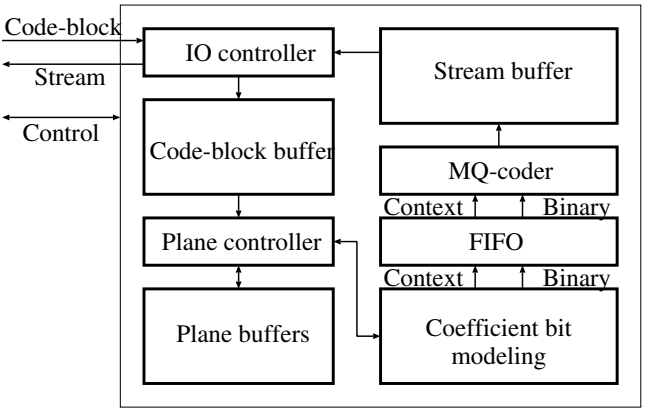

Fig. 5. Architecture of our entropy coder

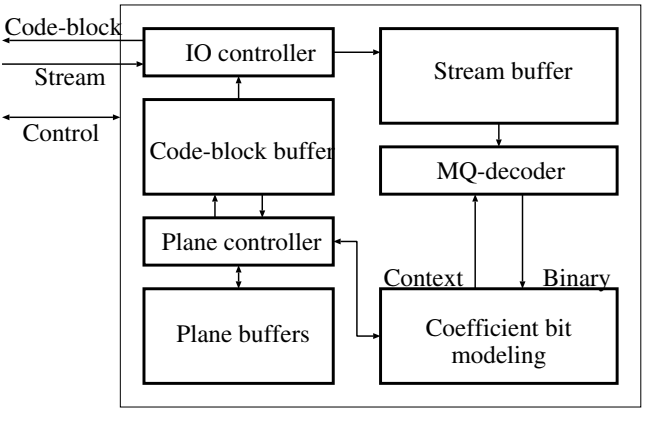

Fig. 6. Architecture of our entropy decoder

The entropy decoder is designed through the use of Verilog-HDL. The result of logic simulation using sample image LENA indicates that when this hardware entropy decoder is employed, the number of cycles needed for entropy coding of whole image is about 5.03 Mcycles, i.e. only 0.283 % of 1776 Mcycles needed for software entropy decoding.

The critical path delay is 7.0 nsec which is concluded by synthesis the entropy decoder with 0.18 μm CMOS technology. The gate counts for this module is 7,901.

Entropy Codec. In our framework, an entropy codec is also prepared in addition of the above mentioned entropy coder and decoder. By sharing some part of circuits between MQ-coder and MQ-decoder and the circuits to generate contexts of coefficient bit modeling for coding and decoding, we can successfully reduce

Table 6. Comparison of the number of gates between entropy coder, decoder and codec

	Submodule	#gate	total
Entropy coder	MQ-coder	2,983	5,658
	Coefficient bit modeling	2,675	
Entropy decoder	MQ-decoder	3,881	6,454
	Coefficient bit modeling	2,645	
Entropy codec	MQ-codec	6,048	9,713
	Coefficient bit modeling	3,665	

Table 7. Main features of the LSI

Technology	Hitachi 0.18μm CMOS
Interconnect	5 metal layers, PolySi
Power supply	1.8V
Chip area	5.9×5.9 mm^2
Design method	Standard-cell-based
Package	256pin BGA

Table 8. The numbers of gates and memory bits

#gate of Xtensa	44,000
#gate of DWT	17,650
#gate of Entropy coder	10,207
#bit of FF in DWT module	24,960
#bit of FF in Entropy coder	6,144
#bit of RAM of Entropy coder	65,536

#gate dose not include the gates of FF used as memory.

Table 9. The number of cycles

function name	total cycle by software	total cycle by using this LSI
`encode`	1326.6 (100%)	163.8 (12%)
`entropy_coding`	887.8 (67%)	3.20 (0.36%)
`FDWT_97`	278.4 (21%)	0.215 (0.08%)

Unit of number of cycles is Mcycle.
% of total cycle by software means the ratio to cycle of `encode`.
% of total cycle by using this LSI means the ratio to cycle by software.

the number of gates of the entropy codec with maintaining its performance. The comparison of the numbers of gates required for the dominant parts, which are MQ-coder/decoder and coefficient bit modeling, among entropy coder, decoder and codec is summarized in Table 6. The number of gates for MQ-codec is 88% of that for the combination of MQ-coder and MQ-decoder, and the number of gates of coefficient bit modeling for codec is 69% of that for the combination of those for encoding and decoding.

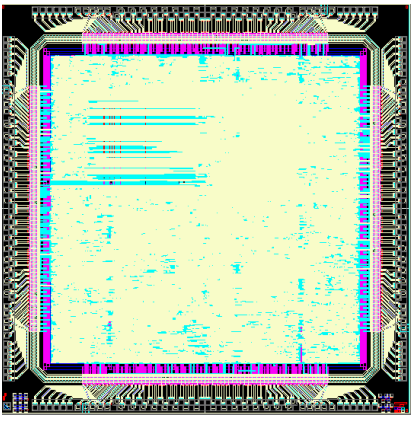

Fig. 7. Layout patterns for the LSI

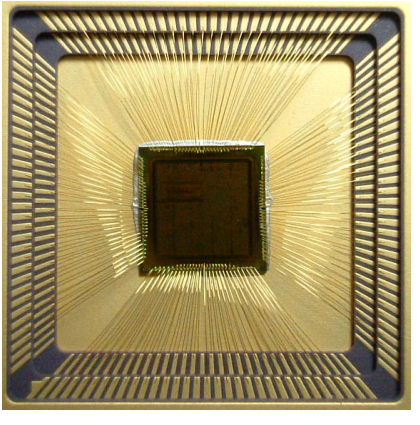

Fig. 8. Photograph of the LSI

6 LSI Implementation Result

In order to demonstrate the practicability of the proposed framework, we fabricated an JPEG2000 encoder LSI, which consists of our DWT hardware module, hardware entropy coder module, and Xtensa. A photograph of the LSI and layout patterns attained for the LSI are shown in Figs. 8 and 7, respectively.

Table 7 summarizes the specifications of the fabricated LSI. The LSI is with 1-Kword $\times$ 32-bit single port RAM as the code-block buffer and stream buffer of the entropy coder. The line buffer of DWT module and the plane buffers of entropy coding module are implemented by flip-flop (FF) arrays. The numbers of gates and memory bits are summarized in Table 8. The critical path delay of this LSI is 18 nsec, which assures 55.5 MHz operation.

The comparison between the number of cycles needed to encode the test image LENA by software and that by using this LSI is summarized in Table 9.

7 Conclusion

In this paper, a novel design framework to realize an efficient implementation of JPEG2000 encoder, decoder, and codec in accordance with the requirements and constraints of each terminals and applications has been proposed. This framework is distinctive in that for each procedure of JPEG2000 coding system, implementation scheme can be selected among software implementation, software implementation accelerated with user-defined instructions, and dedicated hardware implementation, so as to optimize the system organization. To demonstrate the practicability of the framework, we fabricated an LSI to exemplify a generated system implementation, in which our DWT hardware module and hardware entropy coder module were implemented with configurable processor Xtensa.

Acknowledgement. The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration by Hitachi Ltd. and Dai Nippon Printing Corporation.

References

1. ISO/IEC JTC1/SC29/WG1, "Information technology – JPEG2000 image coding system: Core coding system," Oct. 2002.
2. Tensilica, Inc., *Xtensa Application Specific Microprocessor Solutions — Overview Handbook*, Sept. 2000.
3. Tensilica, Inc., *Tensilica Instruction Extension (TIE) Language — User's Guide*, Sept. 2000.
4. ISO/IEC JTC1/SC29/WG1, "JPEG2000 verification model 9.1 (technical description)," June 2001.
5. ISO/IEC JTC1/SC29/WG1, "Draft of FPDRAM-1 to 15444-1," Dec. 2000.
6. Kuan-Fu Chen, Chung-Jr Lian, Hong-Hui Chen, and Liang-Gee Chen, "Analysis and architecture design of EBCOT for JPEG-2000," in *Proc. of the 2001 IEEE International Symposium on Circuits and Systems (ISCAS 2001)*, Vol. 2, pp. 765–768, Mar. 2001.