CAM-Based Huffman Coding Architecture for Real-Time Applications

Takeshi Kumaki, Yasuto Kuroda, Tetsushi Koide and Hans Jürgen Mattausch
Research Center for Nanodevices and Systems, Hiroshima University, 1-4-2, Kagamiyama, Higashi-hiroshima, Hiroshima, 739-8527, Japan

Hideyuki Noda, Katsumi Dosaka, Kazutami Arimoto and Kazunori Saito
System Core Technology Div. System Solution Business Group Renesas Technology Corporation 4-1 Mizuhara, Itami-shi, Hyogo, 664-0005, Japan

INTRODUCTION

Huffman coding is probably the best known and most widely used data compression technique for many software and hardware applications. Generally Huffman coding needs to prepare a code word table that contains the information of mapping between the real data and the code words for encoding. Frequent data is encoded by short code words and less frequent data is encoded by longer code words and thus the compression effect is achieved. This mapping table may be constructed by complete pre-scanning of the real data before coding. However, such a method takes a long processing time and large intermediate storage, so that application is difficult, in particular for real-time encoding. There are two methods for eliminating this pre-scan time in real-time applications. The first method which is called Static Huffman coding is to use a known or default code word table for encoding. The second method which is called Adaptive Huffman coding is to use an encoding tree, which is adaptively constructed and maintained at sender as well as receiver side. While these two methods are often sufficient, they have a number of remaining problems with respect to flexibility and implementation. The first method is difficult to use for real-time applications when the input symbols change their frequency distribution often as for example in the case of video pictures. The second method is hard to implement in hardware and has a negative effect on the compression ratio during phases of construction (start-up) or reconstruction (changing frequency of symbols) of the encoding tree.

For overcoming above problem, a novel architecture for Huffman encoding is proposed. The present paper develops the architecture concept published in [1], [2]. It uses a Content Addressable Memory (CAM) for storing the code word table and enabling a fast Huffman encoding.

PROPOSED ARCHITECTURE

In this section, a CAM-based Huffman coding architecture for real-time applications is proposed as a novel architecture for Huffman encoding. The CAM is exploited to implement the symbol table and to enable fast Huffman encoding in combination with a RAM storing the code word table. At the same time, the code word table is reconstructed according to the frequency of received symbols and is up-dated in real-time. Since the two functions (encoding and reconstructing) work in parallel, the proposed architecture can keep high compression ratio and overcome many problems of conventional Huffman coding. The proposed architecture consists of two functional blocks, encoder and reconstructor. The block-diagram is shown in Fig. 1.

The encoder block is composed of a CAM for the symbol table and a RAM for the code word table and can translate input symbols into the corresponding Huffman code. Actually two code word tables are located in the encoder. One of these tables, the active table generates the code words and the other table, the shadow table prepares an optimized version of the presently used code word table. The switch is used to exchange active and shadow table.

The reconstructor block is composed of an assign module and swap module. The assign module generates an optimized Huffman code according to the most recent frequency distribution of input symbols. The corresponding optimized code word table is sent to the encoder for up-dating the shadow code word table. In case that the compression ratio value satisfies a threshold condition, the swap module generates a table exchange signal to the reconstructor and the assign module. Since active table and shadow table exchange their role in Huffman encoding according to the most recent frequency distribution of input symbols, the proposed architecture can keep a high compression ratio.

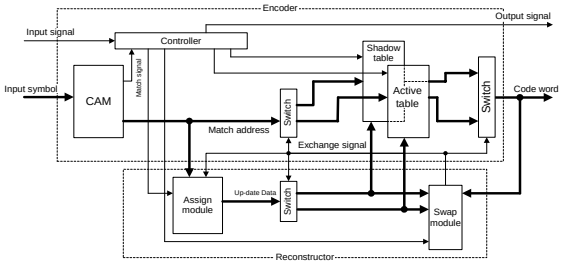


Fig. 1. Block diagram of the proposed architecture.

PROCEDURE OF HUFFMAN ENCODING

The encoder block is composed of a CAM and two SRAMs for active table and shadow table. The principle concept of Huffman encoding is explained in Fig.2. When the CAM receives an input symbol, this symbol is compared in parallel with stored symbols. Then the output port of the CAM sends the determined match address to the corresponding port of the SRAM and finally the code word is outputted. This block can translate input symbols into the corresponding Huffman code. Since a conventional CAM needs just a single-clock-cycle for the comparison process, the proposed architecture can process fast encoding and replace the slower binary trees in Huffman encoding algorithms.

Usually encoding architectures which implement a standardized code word table cannot change the contents in the code word table. However, the proposed architecture constantly updates the contents of a shadow table according to the frequency of receiving input symbols. If the symbol frequency distribution does not fit to the active table anymore, the proposed architecture can exchange the roles of active table and shadow table. The proposed architecture has thus a large flexibility for usage in many applications. Moreover, by preparing the two code word tables, input symbols are always encoded into relatively short code words.

PROCEDURE OF OPTIMIZING THE CODE WORD TABLE

The procedure for optimizing and maintaining the shadow table is shown in Fig.3. This procedure consists of two processes, over-writing process and order-writing process. The over-writing process executes the task of optimizing the code word table. For reducing compressed data size, frequent input symbols have to be assigned to short-bit length code word. After overflowing of an input-symbol the specific counter, up-date address and up-date code word are provided by the assign module. In this example, the code word “00002” is written into the address “0010” in the first step. In the next step, the code word “00023” is written into the address “0000”. Then, the above writing step is repeated till a threshold level

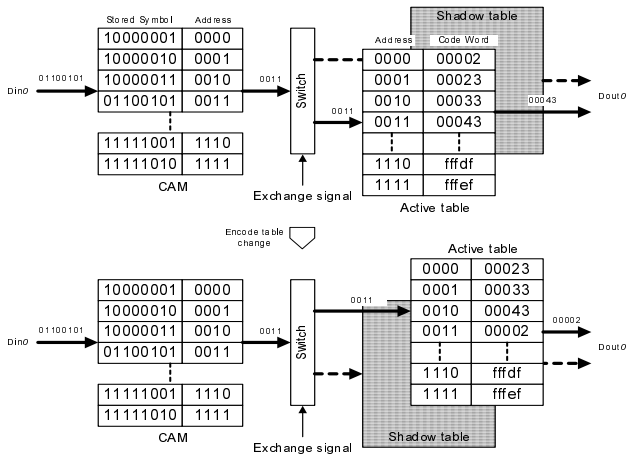


Fig. 2. Procedure of Huffman encoding with a CAM and two code-word tables.

for input-symbol specific counter overflow is reached. The order-writing process executes the task of completing the shadow code word table. After finishing over-writing process, the shadow table has some duplicate data in different addresses. The over-writing process stores the unassigned code words into the not-overwritten addresses for completing the contents of the shadow table. The writing of these code words follows the procedure of the order-writing process. The assign module has a maximum address register, which stores largest address in the over-writing process. In Fig.3, this register stores the maximum address "1110". As a result, the proposed architecture can construct an ideal optimized code word table.

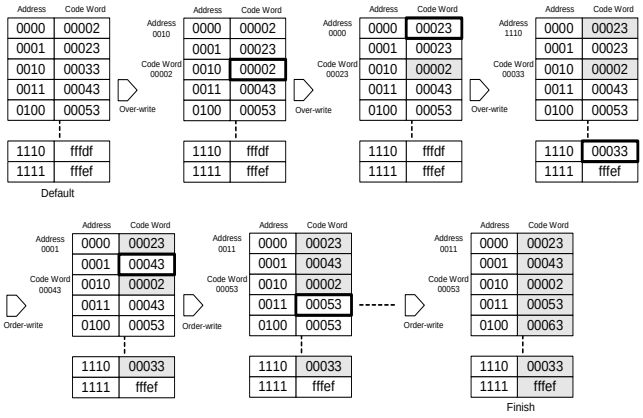


Fig. 3. Procedure of optimizing the shadow code word table.

SIMULATION RESULTS

In this section, simulation results for a JPEG application are presented. The number of compression clock cycles and the size of the compressed picture are evaluated. Table. I shows the comparison results for the four pictures of Fig. 4. Each compressed picture size is evaluated with two different code word tables. Namely a known standardize code word table and an optimized code word table according to the proposed architecture. Four Huffman encoding architectures are compared. Two architectures use a standardized code word table and are distinguished by RAM-based and CAM-based approaches. The third architecture applies an optimized code word table, but is purely RAM-based. The fourth architecture is the proposed architecture. The size of all compressed pictures with optimized code word table is clearly lower than the size of compressed pictures with standardized code word table. Especially, the difference for (D) is very large with about 40%. The picture type (A) often appears in real applications but the difference for (A) is only about 3% in compressed picture size. Nevertheless, it is evident from the picture-size results, that an optimized code word table architecture is very effective for increasing the compression ratio. From the aspect of encoding speed, the RAM-based architecture with optimized code word table needs the largest number of clock cycles, consisting of encoding clock cycles and constructing clock cycles for the optimized code word table. The number of encoding clock cycles is 6 times as many as that of the proposed arcitecture. The number of constructing clock cycles for the optimized code word table increases with picture size, and is about 3,700 times larger than for the proposed arcitecture in case of picture (B). Therefore, it is difficult to use this architecture for encoding in real-time applications such as MPEG. On the other hand, the proposed arcitecture reduces the number of encoding clock cycles to a small fraction and eliminates the constructing clock cycles completely. Thus the proposed architecture appears to be the best solution for encoding in real-time applications.

Fig. 4. Test pictures.

TABLE I
EVALUATED RESULT OF COMPRESSION SIZE AND CLOCK CYCLES.

		(A) Lenna				(B) Dawn of the sun			
Table		standardized		optimized		standardized		optimized	
Architecture		SRAM	CAM	Processor	CHRC	SRAM	CAM	Processor	CHRC
Number of clock cycles		30	5	1,950	5	30	5	18,452	5
Picture size [byte]	Original	246,840				2,359,350			
	JPEG	40,209		39,017	39,075	44,599		35,502	35,597

		(C) Texture				(D) Test screen			
Table		standardized		optimized		standardized		optimized	
Architecture		SRAM	CAM	Processor	CHRC	SRAM	CAM	Processor	CHRC
Number of clock cycles		30	5	593	5	30	5	2,425	5
Picture size [byte]	Original	73,782				308,278			
	JPEG	2,380		1,961	1,973	11,274		6,750	6,815

CONCLUSION

This paper presented a CAM-based Huffman coding architecture for real-time applications as a novel architecture for efficient Huffman encoding. A CAM is exploited to implement fast Huffman encoding, while an optimized code word table is reconstructed and up-dated in real-time. It becomes possible to compress in real-time, keep high compression ratios and apply the proposed architecture to many compression situations flexibly. Consequently, the proposed architecture has many realization possibilities and can be used in hardware for real-time applications.

ACKNOWLEDGMENT

Part of this work has been supported by the 21st century COE program, Ministry of Education, Culture, Sports, Science and Technology, Japanese goverment and a Grant-in-Aid for JSPS Fellows, 175303, 2005.

REFERENCES

[1] T. Kumaki, Y. Kuroda, T. Koide, H. J. Mattausch, H. Noda, K. Dosaka, K. Arimoto, and K. Saito, "CAM-based VLSI architecture for Huffman coding with real-time optimization of the code word table," *Proc. IEEE International Symposium on Circuits And Systems (ISCAS'05)*, pp. 5202–5205, May 2005.

[2] ——, "Multi-port CAM based VLSI architecture for Huffman coding with real-time optimizede code word table," *Proc. IEEE International Midwest Symposium on Circuits And Systems (MWSCAS'05)*, Aug. 2005.

CAM-Based Huffman Coding Architecture for Real-time Applications

Takeshi Kumaki, Yasuto Kuroda, Masakatsu Ishizaki, Tetsushi Koide and Hans Jürgen Mattausch

Research Center for Nanodevices and Systems, Hiroshima University

Hideyuki Noda, Katsumi Dosaka, Kazutami Arimoto and Kazunori Saito

System Core Technology Div., System Solution Business Group, Renesas Technology Corporation

Hiroshima University

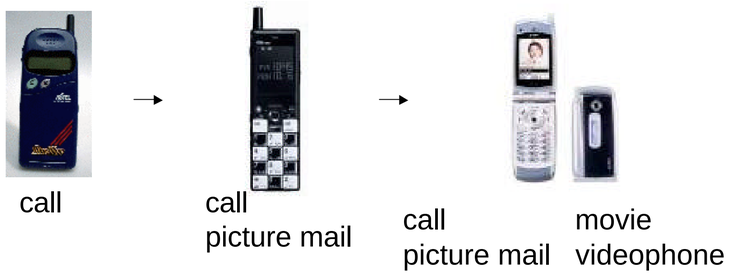
Background & Motivation

- Infrastructure
  - Improving network bandwidth, speed and quality

    (*ex*) Dial (56Kbps) → ISDN (64/128Kbps) → ADSL (1.5/6Mbps) → FTTH (10～100Mbps)

  - Development of mobile devices

    (*ex*) mobile phone



Consumer Needs
- High speed
- High quality

(*ex*) Sending and receiving much contents (data, picture, movie etc.)

Key Points:
- Small data size
- Low processing time
- Losslessness

Solution:
Effective compression technique

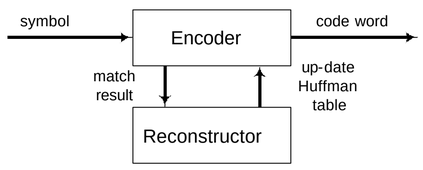
Huffman coding

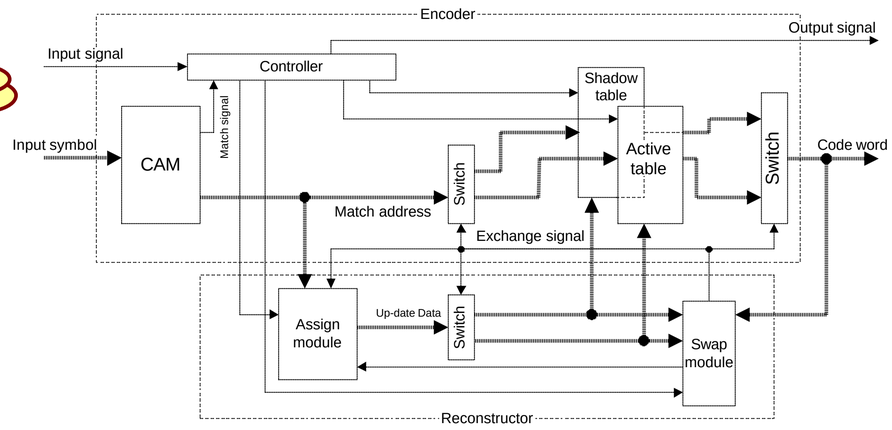
Proposed Architecture for Huffman Coding

- **The CAM-Based Huffman coding Architecture for Real-time Applications**
- Proposed architecture realizes
  - **faster** encoding
  - **up-dating** of optimized code word table in **real-time**

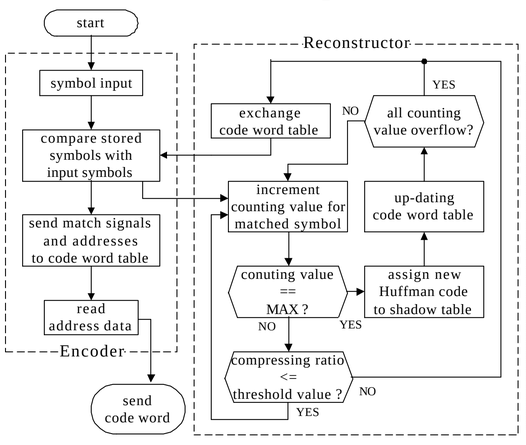
Parallel processing

- Proposed architecture consists of three functional blocks

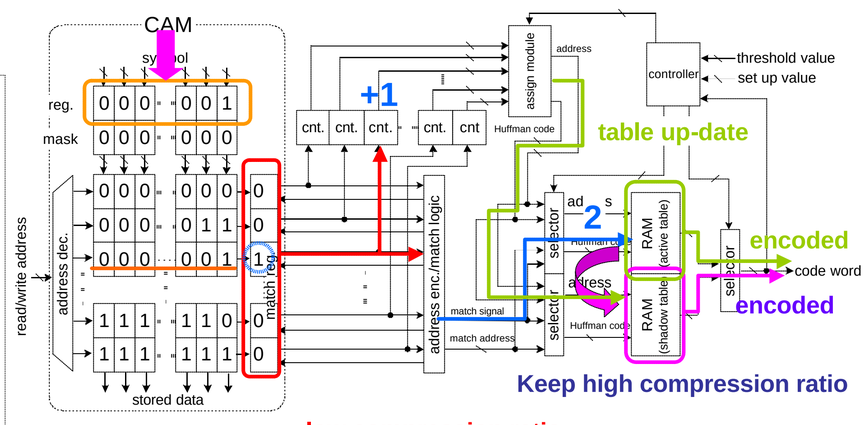




Coding Flow



- Input symbol
- Compare, sending match signal to reconstractor and sending match address to RAM
- - Generate encoded data
  - Increment counter value
- Up-date code word table
- Change code word table



low compression ratio...

Fast Huffman encoding

Huffman Coding

- Most widely used lossless compression technique
- Exploited for many software and hardware applications
  - JPEG, MPEG, ZIP, LHA, MP3, etc.
  - Implemented in many ASICs for digital camera, mobile phone, etc.

Basic Concept **Frequent data is encoded by short code word**

ATAACCCCGG (A x 3, T x 1, C x 4, G x 2)
→10 111 10 10 0 0 0 0 110 110 (A: *10*, T: *111*. C: *0*, G: *110*)

1011110100000110110 → ATAACCCCGG (uniquely decoded)

The Problems of Huffman Coding

- There are two ways to implement Huffman coding

(a) Static Huffman coding

(*Original idea*) Iteratively build a binary tree according to frequency distribution of symbols

This algorithm needs two times scanning of the same data

(*Problem*) **Real-time encoding becomes difficult**

(*Common technique*) **Using standardized code word table**

(*Problem*) **Compression ratio is decreased if symbol frequency distribution does not fit to standardized code word table**

(b) Adaptive Huffman coding

(*Original idea*) Binary tree for coding is created on the fly

(*Problem*) **Complexity in hardware implementation**

Related Works

- Standardized code word table has been embedded in conventional architectures.

Comparison with different architectures capability

Factor \ Architecture	Processor (Software)	SRAM	PLA (Programmable Logic Array)	CAM
Speed	××	×	△	○
Hardware amount	×	△	×	△
Flexibility (∗)	○	×	×	×
Data size	○	△	△	△

Poor

○: good △: medium ×: poor ××: bad

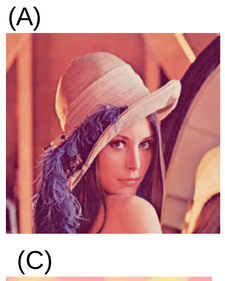
(∗) Keeping high compression ratio in spite of frequency distribution of symbols

Simulation Result for JPEG Application

Test pictures

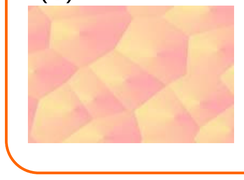
The number of compression clock cycles and the size of compressed picture are evaluated

(A)

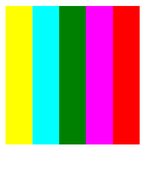


(B)

(C)



(D)



		(A) Lenna				(B) Dawn the sun			
Table		standardized		optimized		standardized		optimized	
Architecture		SRAM	CAM	Processor	Proposed Architecture	SRAM	CAM	Processor	Proposed Architecture
Number of clock cycles		30	5	1,950	5	30	5	18,452	5
Picture size	Original [byte]	246,840				2,359,350			
	JPEG [byte]	40,209		39,017	39,075	44,599		35,502	35,597

		(C) texture				(D) test screen			
Table		standardized		optimized		standardized		optimized	
Architecture		SRAM	CAM	Processor	Proposed Architecture	SRAM	CAM	Processor	Proposed Architecture
Number of clock cycles		30	5	593	5	30	5	2,425	5
Picture size	Original [byte]	73,782				308,278			
	JPEG [byte]	2,380		1,961	1,973	11,274		6,750	6,815

Factor \ Architecture	Processor (Software)	SRAM	PLA (Programmable Logic Array)	CAM	Proposed Architecture
Speed	××	×	△	○	○
Hardware amount	×	△	×	△	△
Flexibility (∗)	○	×	×	×	○
Data size	○	△	△	△	○

○: good △: medium ×: poor ××: bad

- Compressed picture size of proposed architecture is lower than for other standardized compression architectures.
- The number of clock cycles needed in proposed architecture is much lower than for other architectures.

Proposed architecture is the best solution for Huffman encoding

Conclusion

- The CAM-Based Huffman coding Architecture for Real-time Applications is proposed as a novel architecture for Huffman encoding
- Effectiveness of Proposed architecture is verified concerning JPEG application

Future works

- Improving encoding capability: **sequential → parallel**
- JPEG, MPEG: combine proposed architecture with SIMD architecture
- Layout design for VLSI chip