EFFICIENT DECODING OF VARIABLE-LENGTH ENCODED IMAGE DATA ON THE NIOS II SOFT-CORE PROCESSOR

Peter Mårtensson, Jens Persson, Shang Xue, and Bengt Oelmann
Mid Sweden University, Department of Information Technology and Media
SE-851 70 Sundsvall, SWEDEN
Bengt.Oelmann@mh.se

ABSTRACT

This paper presents two approaches to acceleration of variable-length decoding of run-length coded image data in the Nios II embedded processor for Altera FPGA implementation by using customized instructions. In the first approach we examine the possibilities of accelerating variable-length decoding for the JPEG standard by implementing the code table look-up as a customized instruction. In the second approach we replace the VLC code table with a coding technique we have previously proposed called *Alternating Coding* and implement customized instructions for accelerating the decoding. The VLC decoder is implemented in software using the C programming language with function calls accessing the hardware accelerated instructions. For decoding using standard JPEG VLC codes, one customized instruction resulted in a speed-up of 3.7 times compared to an all software implementation. For the *Alternating Coding* VLC, the software and hardware accelerated implementations both resulted in a speed-up of more than 3.5 times compared to the standard JPEG software implementation.

KEYWORDS

VLC decoding, FPGA, soft-core processors

1. INTRODUCTION

Entropy coding using Variable-Length Codes (VLC), is a widely used lossless compression technique for image and video data, such as the run-length coded images. It is used in many video and image compression standards such as JPEG and MPEG-2. Compression is achieved by assigning shorter codewords to symbols with lower probabilities and longer codewords to those with lower probabilities. The variable length of the codewords makes the decoding into a highly sequential process. In order to decode a codeword, the previous codewords must be decoded in sequence in order to determine where in the input buffer the codeword starts. These data-dependencies limit the throughput of VLC decoders and parallelization is difficult to achieve. Even though VLC decoding is not a computationally intensive part of the image or video decoder, it still occupies a significant part of the processor's time.

Different types of processor architectures supporting efficient implementation of multimedia applications ranges from fully custom to general purpose processors (GPP) with support for multimedia that are fully programmable. The designer's choice among these is most often based on trade-offs between flexibility and performance. For system integration on FPGA these processor architectures are not today available to the designer. The options are to design dedicated hardware for the entire image/video decoder, use a soft-core GPP without any multimedia support, or a mix; with functions in hardware used as co-processors to the GPP.

The Nios II from Altera [1] is a configurable soft-core GPP with the possibility of adding customized instructions executed in dedicated hardware to accelerate time-critical parts of software algorithms. In this paper we examine the possibilities of improving the VLC decoding throughput by adding application-specific instructions. Other tasks in image or video decoder such as color-space transformations, IDCT (Inverse Discrete Cosine Transform), motion compensation, can be efficiently implemented through parallelization by adding sufficient hardware resources for the custom instructions. To study this we implement a JPEG image decoder but our results can also be applied to, for example, MPEG-2 since the VLC decoding procedures and

the statistical models of the data are very similar. We start with a JPEG decoder implemented in the C programming language and analyze the execution times of the tasks of the VLC decoding. After identifying the most time-consuming parts, they are implemented in hardware in such a way that they could be accessed through custom instructions and used through function calls from C.
Two approaches to high-throughput VLC decoding have been examined. First we look at how to speed-up decoding when the standardized VLC code table is used. Then we investigate a VLC coding method, we call alternating coding (ALT-coding), for JPEG that is constructed for efficient implementation. Our results show that by simply having the table look-up as a customized instruction for the standard JPEG VLC results in significant improvement. By using the ALT-coding, we show that the Nios II architecture perform decoding in software as efficiently as the hardware accelerated standard JPEG decoder.

2. IMPLEMENTATION OF VLC DECODING

2.1 Target implementation architecture

The Nios II architecture allows incorporation of custom hardware that can be accessed from software. The custom logic, see Figure 1, takes two 32-bit words from registers in the processors register file. The custom logic returns a 32-bit word.

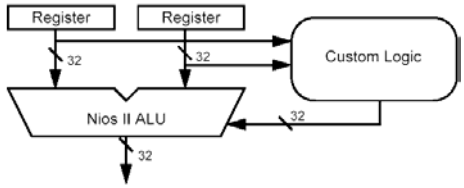

Figure 1. Nios II Datapath

2.2 Implementation of standardized VLC decoding

VLC decoding is performed by reading bits from a data stream from memory and by matching a vector of bits, which can be of variable length, the variable-length codeword is found in a look-up table. In JPEG the data format is as shown in Figure 1a. Four look-up tables are used for the different DC and AC components; AC/DC-luminance and AC/DC chrominance DCT coefficients. The DCT coefficients are run-length encoded and a simplified description of the format is given in Figure 1b. The *code_word_length* and *additional_bits* are both of variable length. When decoding, first the *code_word_length* is determined through the look-up tables. If AC-coefficients are decoded it contains information about the length of the following *additional_bits* field and the run-lengh. If DC-coefficients are decoded, it will together with the *additional_bits* form the DC-value.

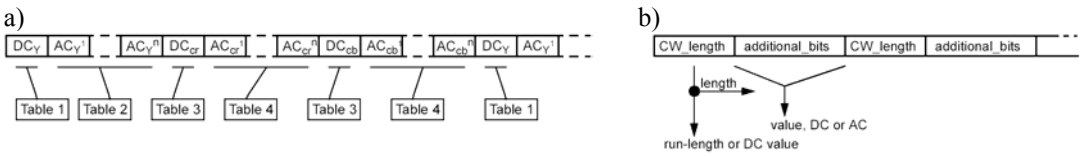

Figure 1. Format for coded image data. a) DCT coefficients and b) Run-length coded DCT coefficients

The average codeword length in a JPEG image is between 3 to 4 bits. When searching for a valid codeword in the data stream a considerable amount of time is spent in the loop accessing the tables before a valid codeword has been found. Also handling the input data stream with variable data lengths is time-consuming. In order to accelerate both the look-up and the buffer handling a customized instruction was

implemented to support these two tasks. The associated hardware is four combinational logic modules where each of them is a modified VLC look-up table which performs one codeword decoding per clock cycle. The input is 32-bit VLC encoded image data. The instruction returns a set of data consisting of codeword length, number of additional bits, and the run-length, all stuffed in a 32-bit string returned to software. The custom-instruction function call from C is:

{ code_word_length, number_of_additional_bits, run_length_value } = custom_look_up_table(VLC);

The major advantage of the hardware-accelerated instruction, compared to the pure software, is that it eliminates the loop procedure repeatedly checking look-up tables for valid codewords. As a consequence, the number of clock cycles needed for codeword decoding becomes constant and independent of the codeword length. The handling of the input data buffers also becomes simpler when the codeword length and the number of additional bits are returned by the custom instruction which makes it possible to extract the additional bits immediately from the VLC string.

2.3 Implementation of ALT-coded VLC decoding

The Alternating Coding (ALT-Coding) that we previously have proposed for various commonly used VLCs, e.g. in [2], enables simple decoder structures resulting in high speed performance and low power consumption for dedicated hardware implementations. In this work we apply the ALT-coding method to a class of VLC codes that is called *UVLC* (Universal Variable Length Code). The UVLC was included in the video coding standard H.26L and belongs to a VLC family named *EG* (exp-Golomb) codes. The EG codes have shown to be very efficient in coding of the image and video data and are still included in current video coding standards. While the ALT coding of the UVLC is one example, it can be easily extended to the efficient encoding of entire EG family.

The general idea of ALT-coding is to structure the code is such way that codeword (CW) boundary detection is simplified. In the case of UVLC, the code is constructed by *Even Indexed Bits* (EIB) and *Odd Indexed Bits* (OIB). By ALT-encoding the UVLC the EIBs are separated from the OIBs, see Figure 2. For the OIBs, two codeword tables are applied, one is {1, 11, 111, 1111 …} and the other is {0, 00, 000, 0000 …}. They are alternated in the encoding procedure so that the codeword boundaries and thereby the codeword lengths can be easily determined by detecting the value changes in the OIB sequence. The EIBs can be of any binary combination and their code lengths are computed on the basis of length of the corresponding OIB. For the details on the construction of code tables for ALT-coded UVLC, we refer to [2].

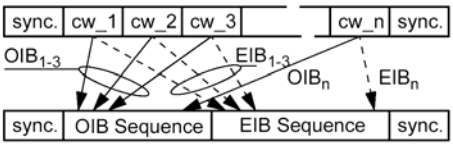

Figure 2. Construction of an ALT-coded UVLC packet

By using ALT-coded UVLC we have the advantages of simple codeword boundary detection and computation of the values instead of table look-up. The boundary detection loop, detecting 0→1 or 1→0 transition in the data stream, is used to establish the codeword length from the OIB sequence. When the length is determined, the corresponding EIB bits are directly extracted and the value can be computed. All decoding tasks; the codeword length extraction, value computation, and input buffer handling, are equally time-consuming. Three custom instructions have been implemented. The first, called CW_length, extracts the codeword length from the OIB sequence. The second, called RL_computation, computes the run-length, and the third, called value_computation, computes the value. The custom-instruction function calls from C are:

code_word_length = CW_Length(IOB_sequence);
run_length = RL_computation(codeword);
value = value_computation(codeword, sign_bit);

As for the hardware accelerated standard JPEG decoder, the hardware accelerated ALT-decoder eliminates the boundary detection loop which results in a reduction in number of clock cycles needed to a constant,

independent of the codeword length. In addition, the computation and mapping of the run-length and value were implemented as custom instructions to further reduce the number of clock cycles.

3. EXPERIMENTAL RESULTS

For our experiments we have implemented the 32-bit Nios II soft-core processor with our hardware accelerated custom instructions using the Quartus II software. The processor was implemented in the Cyclon device EPIC20F400C7 on an Altera development board. The Table I below shows the decoding performance for software implementations of the standard and the ALT coded JPEG and their accelerated counterparts and two related works on codeword sequential decoding on programmable processor architectures.

Table 1. VLC decoding throughput.

VLC Decoder	Cycles / codeword		Cycles / coefficient			
	Nios II SW	Nios II HW	Nios II SW	Nios II HW	[4]	[5]
Standard JPEG	783	164	56.0	14.9	16.9	26.0
ALT-coded JPEG	212	141	15.9	13.8	-	-

The hardware implementation costs for the customized instructions are given in Table 2. The Nios II with accelerating hardware have an increase of 13% in number of LUTs and the accelerated ALT-JPEG has a corresponding increase of only 4%.

Table 2. Implementation costs for additional custom instructions.

	Nios II	Nios II with std. JPEG acc.	Nios II with ALT-JPEG acc.
# LUTs	4708	5313	4908
# Registers	2226	2229	2228

4. CONCLUSION

In comparison with implementations on other processor architectures, the VLC decoder on the Nios II processor is competitive for image and video applications. For the standard JPEG VLC decoding we have shown that by just adding one customized instruction for the codeword table look-up, a significant improvement has been achieved. In contrast, the ALT-decoder can be efficiently implemented directly on the Nios II architecture in software with performance close to the hardware accelerated standard JPEG. For future work parallelization VLC decoding will be investigated, where we foresee that the ALT-codes are suitable for parallel VLC decoding. We also believe that the other tasks (IDCT, motion compensation etc.) can be efficiently implemented on the Nios II architecture since 64 bits can passed to the hardware modules.

REFERENCES

1. Altera, *http://www.altera.com/*
2. S. Xue and B. Oelmann, 2003. Alternating Coding for Universal Variable Length Code, *IEEE International Conference on Image Processing*, Barcelona, Spain.
3. S. Xue and B. Oelmann, 2003. A Coding Method for UVLC Targeting Efficient Decoder Architecture, *IEEE International Symposium on Image and Signal Processing and Analysis*, Rome, Italy.
4. M. Simat et al. 2002. Entropy Decoding on Trimedia/CPU64, *International Workshop on Systems, Architectures, Modeling and Simulation*, Samos, Greece.
5. F. Bonomini et al. 1996. Implementing an MPEG2 Video Decoder Based on the TMS320C80 MVP, *Application Report SPRA332*, Paris, France.