Parallel Variable-Length Decoder Architecture for Alternated Coded GR-Codes

Shang Xue and Bengt Oelmann

Department of Information Technology and Media, Mid Sweden University, SE-851 70 Sundsvall, Sweden.
E-mail: Xue.Shang@mh.se, Bengt.Oelmann@mh.se

Abstract -- In this paper we present a decoder architecture for variable-length codes capable of decoding alternated coded Golomb-Rice (GR) Codes in parallel. An architecture developed specifically for GR-codes and a coding method, we call alternating coding, simplify the decoder structure and enable parallel decoding. The proposed decoder detects the codeword lengths for all codewords in the input buffer in parallel. The logic delay for this is not dependent on the input buffer size which makes it possible to detect an arbitrarily large number of codeword lengths at a constant speed. The decoding speed is however limited by the output stage that outputs a serial stream of decoded symbols. Estimates of the speed performance indicate that for an ASIC implementation the maximum throughput is more than 800 MSymbols/s and for an FPGA implementation more than 300 MSymbols/s.

1. INTRODUCTION

The most used entropy coding techniques include Variable-Length Coding (VLC) and are used in image and video coding standards like JPEG and MPEG. Compression is achieved by assigning shorter codewords to symbols with high probability and longer codewords to symbols with lower probability. The variable length of the codewords makes the decoding to a 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. This data-dependency is limiting the throughput of the VLC decoders. The conventional VLC decoder is implemented as a Finite-State Machine (FSM) where the input rate is constant one bit per clock cycle and the output rate is variable [1]. This is a compact solution but the decoding speed is very low. Another common solution provides constant output rate of one symbol per clock cycle [2]. The major drawback here is the relatively large critical timing path in the feedback loop that includes Barrel-Shifter, codeword table (ROM/PLA), and accumulator. This feedback path cannot be pipelined and will therefore limit the performance of this architecture. In a recently presented work [3] a decoder performing parallel extraction of the codeword lengths is presented. Here the problem of the recursively dependency has been greatly reduced but not been removed.

In all previous research on designing high-throughput VLC decoders, general decoders able to decode any variable-length code have been considered. For video and image coding certain types of codes have shown to be suitable. For example, in H.26L the Exponential-Golomb code and for lossless image compression in JPEG-LS GR-codes is suggested. Since the applications of VLC today are heavily dominated by image and video coding, we find it motivated to focus on finding better solutions for the specific codes used in these applications to the price of loss of generality.

In this work we focus on designing a high-throughput decoder for GR-codes. Joint design of hardware architecture, format of the code, and format of the VLC packet gives us new possibilities for high-level optimizations. In summary, there are two important steps in our approach that enables parallel codeword detection. Firstly, the GR-code is split into a prefix- and a suffix part. Secondly, the prefix of the code is of variable length and its bit pattern is coded in, what we call, an alternating way to simplify the codeword boundary detection. The suffix, which is a fixed-length code, does not need any special function for codeword boundary detection.

We target a decoder architecture with constant input rate and variable output rate. It decodes all codewords in an arbitrarily large input buffer in parallel. The output of the symbols comes serially in a variable rate and can be fed to an external FIFO buffer. The overall architecture is shown in Figure 1.

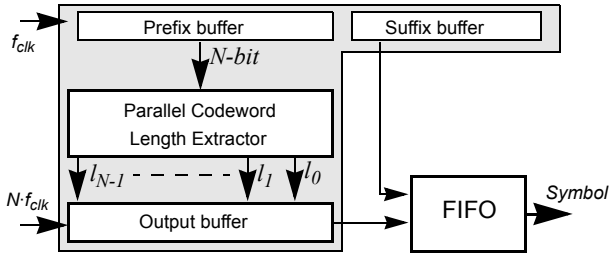

Fig. 1: Overall decoder architecture

Even though it can decode an arbitrarily large input buffer with constant delay, the throughput has an upper limit which is defined by the maximum speed the output buffer can deliver a serial sequence of symbols. The decoder is implemented in RT-level VHDL and from the synthesis results, the maximum throughput is found to be more than 300 MSymbols/s and 800 MSymbols/s for FPGA and ASIC implementations respectively.

2. ALTERNATING CODING

Golomb-Rice codes were first introduced in [4] and [5]. The codeword table in Table 1 illustrates how the GR-code is constructed. The GR-code is parameterizable with the parameter k and is composed of a prefix code of variable length and a fixed length suffix of length k.

	k=1				k=2			
	Golomb -Rice		Alt-Coded GR		Golomb-Rice		Alt-Coded GR	
	Prefix	Suffix	Prefix	Suffix	Prefix	Suffix	Prefix	Suffix
0	0	0	0/1	0	0	00	0/1	00
1	0	1	0/1	1	0	01	0/1	01
2	10	0	00/11	0	0	10	0/1	10
3	10	1	00/11	1	0	11	0/1	11
4	110	0	000/ 111	0	10	00	00/11	00

Table 1. Codeword table

In order to simplify the codeword boundary detection, the prefix codes are modified by, what we call, *alternating coding* (alt-coding). In a sequence of codes the prefixes are coded alternating with all-zeros and all-ones codes. In an alt-coded VLC packet, the prefixes and suffixes are separated where the suffixes for all codewords are located in the beginning and all suffixes are in the end of the packet, see Figure 2.

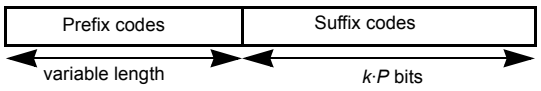

Fig. 2: Alt-coded VLC packet

When the total number of codewords, P, in a packet is known, the $k \cdot P$ last bits are identified as suffix codes and the rest as prefix codes. To decode the symbol, the lengths of the prefix codes can be decoded separately and then simply concatenated with their respective suffix code.

3. DECODER ARCHITECTURE

For alt-coded GR-codes the decoding is reduced to length extraction of the prefix codes. In this section a parallel architecture for length extraction is presented.

Each clock cycle of f_{clk} the decoder takes in a new set of codewords of N bits to the *Prefix Buffer*. The lengths of the prefixes are extracted in parallel by the *Parallel Codeword Length Extractor*. At most, when all codewords are of minimum length of one bit, N codewords are decoded. The output buffer is therefore designed to receive N codeword lengths (l_{N-1} to l_0). For normal image data the codeword lengths are distributed within the range of one to maximum prefix codeword length M. This means that normally, not all positions in the output buffer will be occupied. An empty-indicator, called e_i, is generated and shows whether a buffer position is empty or occupied. The decoded codeword lengths are serially put on the outputs of the *Output Buffer* which is a Parallel-Input Serial Output (PISO) register. The maximum output rate is when the prefix codeword lengths are all of one bit which makes it necessary to clock the *Output Buffer* at $N \cdot f_{clk}$. The empty-indicator from the *Output Buffer* is used for indicating the existence of data out from the Output Buffer. The Suffix Buffer is a PISO where shifting of k steps takes place when e_i is true.

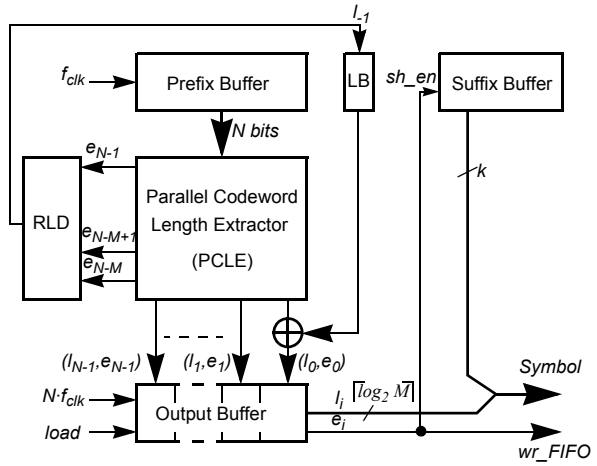

Fig. 3: Detailed decoder architecture

Under the condition that the codeword length extraction can be parallelized, the critical timing path in this architecture is in the *Output Buffer*: $t_{critical} = t_{mux} + t_{DFF}$, where t_{mux} is the delay in a 2-1 multiplexer and t_{DFF} is the delay in a D-flip/flop.

Before going in details on how the proposed *Parallel Codeword Length Extractor* (PCLE) is designed, an example is presented showing the working principle in Figure 4. The input buffer contains the alt-coded prefixes, of maximum length of four bits ($M=4$), in the vector C. The rightmost bit in the buffer is considered to be the first bit. From C the boundary vector B is computed where a '1' indicates the position of the last bit in a prefix code. The length extraction is segmented to windows of M bits. Based on the first four bits ($i=0$) in the B vector, the first occurrence of a boundary, i.e. a '1' at position 0 gives us the length $l_0=0$. It is guaranteed that the shortest prefix, that is one bit long, is extracted in this window. The next window can therefore be positioned one bit left to the previous window. In general, there will be N M-bit windows for a prefix buffer of N bits. In the next window ($i=1$) a boundary is found at position 3 ($l_1=3$). This boundary is also found in the windows $i=2$, 3, and 4. These kind of boundaries must be disabled by providing an offset for extracting the next length. In window $i=2$, the length extraction starts where the previous code ends. Here the

situation occurs that no boundary is found.

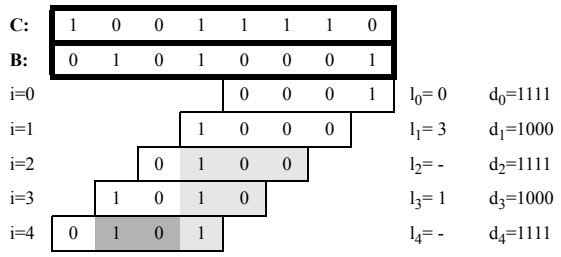

Fig. 4: Example of parallel length extraction

In order to achieve parallel length extraction, each window has a *Length Extraction* (LE) unit. It contains three functions: 1) length extraction function providing the prefix length (l_i), 2) computation of the disable mask (d_i) that is fed to the following LE-units and 3) computation of the empty-indicator e_i. The offset is computed on the basis on B exclusively. The length is based on B and the offsets from the *M-1* previous LE-units. The block diagram of the PLCE is shown in Figure 5. The delay in a parallel architecture cannot be dependent on the size of the prefix input buffer (N). For the proposed architecture the delay is dependent on the maximum codeword length (M) and not N which allows unlimited parallelization.

In the LE-unit the offset D_i, based on the disable masks from the *M-1* previous LE-units, is computed as:

$$D_i = \bigwedge_{j=i-M+1}^{i-1} shr_j(d_j)$$

Where the functions $shr_j(d)$ shifts d j positions right with '1' shifted in from the left. When implemented, this is done by wiring. The prefix codeword length is computed as: $l_i = length(D_i \wedge C_i) - length(D_i)$.

Where C_i is the prefix code for the i:th window and the length function is returning the position of the first occurrence of a '1' from the right in the vector.

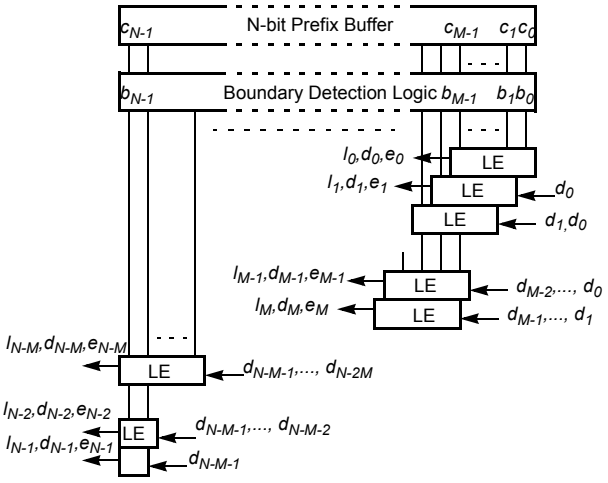

Fig. 5: Parallel codeword length extraction

The empty-indicator is computed as: $e_i = \neg(D_i \wedge C_i)$

Critical timing path comes from computing the length l_i and is implemented as shown in Figure 6.

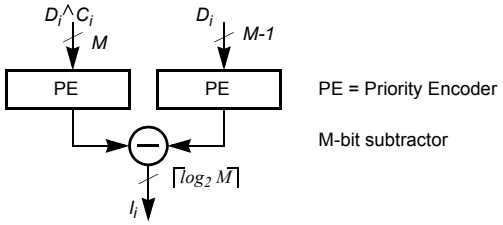

Fig. 6: Codeword length detection unit

It may occur that only the first part of the last prefix code resides in the *Prefix Buffer* and the rest will be loaded the next clock cycle. The function *Remaining Length Detector* (RLD), shown in Figure 1, decodes the length of the partial code from the *M-1* empty-indicators and it is stored in the *Length Buffer* (LB) to be used for the next set of data loaded in the *Prefix Buffer*.

The alternating coding enables simple logic for the length extraction. This is important, even though the architecture can be parallelized without limitations, it will affect the required clock frequency of the output buffer.

4. ASIC AND FPGA IMPLEMENTATION

This section presents the performances and area-costs of the parallel decoder implementation to both an ASIC and an FPGA technology.

The computational logic for the PLCE and the RLD are implemented in RT-level VHDL. Logic synthesis using Synopsys' Design Compiler for the ASIC implementation in a 0.5 µm CMOS technology and WebPack for the FPGA implementation in Xilinx's Spartan IIe device. The delays have been obtained from pre-layout timing analysis with wire-load models from the silicon vendors.

When designing a decoder with maximum throughput, the minimum size of the *Prefix Buffer* is determined by the maximum clock frequency of the *Output Buffer*. This will determine the number of LE-units that is equal to the number of bits in the *Prefix Buffer*. The number of LE-units required for maximum decoding throughput is:

$$N_{LE} = \left\lceil \frac{t_{LE} + t_{RLD}}{1/f_{outbuff}} \right\rceil = N$$

Where t_{LE} is the delay of one LE-unit and t_{RLD} is the delay of the RLD. The delays t_{LE} and t_{RLD} are dependent of the maximum prefix codeword length M. Decoders for maximum codeword lengths of 4, 8, 12, 16, 24 and 32 have been designed and evaluated. The suffix length (k) does not affect the computational logic and different values of k are therefore not investigated. In Figure 7 the number of parallel LE-units required for maximum

throughput is shown. Maximum throughput for the ASIC and FPGA implementations are 810 MSymbols/s and 340 Msymbols/s respectively. For large values of *M*, the FPGA requires more LE-units compared to the ASIC implementation to reach maximum throughput. The main reason for this is the larger increase in wire delays for the FPGA.

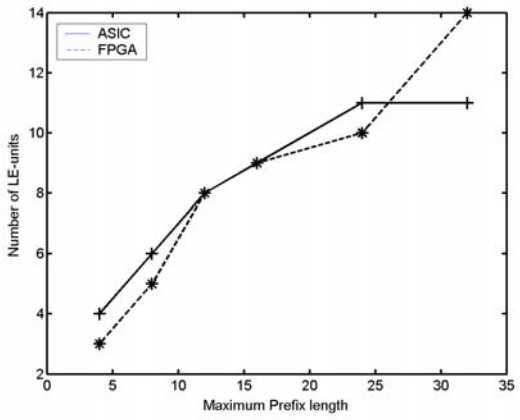

Fig. 7: Number of parallel LEs for maximum throughput

The area required for ASIC and FPGA implementations of the decoder for different values of *M* are shown in Figure 8. The area grows linearly for the ASIC implementation. For the FPGA implementation the area increases rapidly for *M=32* because the delay of the LE-unit is large and must be compensated by increasing the parallelity that requires more LE-units.

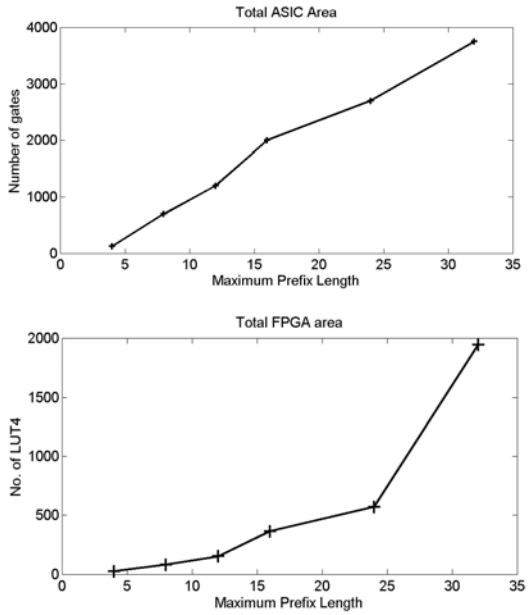

Fig. 8: Area for computational logic

5. CONCLUSIONS

The maximum speed of the parallel decoder for GR-codes, which is proposed in this paper, is limited by the *Output Buffer* that is implemented with a PISO register. It is shown that by using prefix-suffix separation together with alternating coding, simple codeword length extraction is enabled. The main contribution of this paper is that it shows that the recursive dependency between codewords has been removed and input buffers of arbitrary size can be decoded at constant delay. The bottleneck that is in the Output Buffer is still a problem that requires further investigations.

We have shown that there are room for improvements of Variable-Length decoding by doing code-specific optimizations. The alternating coding technique, that we are proposing, has been applied to different VLCs to improve the speed, area, and power performances of the decoders [6, 7, 8] as well as the error-resiliency [9, 10].

6. REFERENCES

[1] M. Mukherjee et.al, “Efficient VLSI designs for data transformation of tree-based codes,” *IEEE Trans. on Circuits Syst*., vol. 38, pp. 306-314, 1991.

[2] S.M Lei and M.T. Sun, “An entropy coding system for HDTV applications,” *IEEE Trans. Circuits Syst. Video Technol.*, vol.1, pp. 147-155, 1991.

[3] J. Nikara et al., “Parallel Multiple-Symbol Variable-Length Decoding,” Proc. ICCD’02, 2002.

[4] S. W. Golomb, ”Run-length endodings,” in *IEEE. Trans. Inf. Theory*, vol. IT-12, pp. 399-401, July 1966.

[5] R. F. Rice, ”Some practical universal noiseless coding techniques,” in *Tech. Rep.*, JPL-79-22, Jet Propulsion Laboratory, Pasadena, CA, March 1979.

[6] S. Xue and B. Oelmann, "Efficient VLSI implementation of a VLC Decoder for Universal Variable Length Code using alternating coding," *IEEE Annual Symposium on VLSI*, Tampa, Florida, USA, 20-21 February, 2003.

[7] S. Xue and B. Oelmann, "A coding method for UVLC targeting efficient decoder architecture," to *3rd IEEE International Symposium on Image and Signal Processing and Analysis*, September 18-20, 2003, Rome.

[8] S. Xue and B. Oelmann, “Efficient VLSI implementation of VLC decoder for Golomb-Rice code using alternationg coding, in manuscript, 2003.

[9] S. Xue and B. Oelmann, "Alternating Coding for Universal Variable Length Code," *IEEE International Conference on Image Processing*, September 14-17, 2003, Barcelona.

[10] S. Xue and B. Oelmann, "Error Resilient coding of DCT coefficients using alternating coding of UVLC," Norsig-2003.