Proceedings of ICCD 2002, Freiburg, Germany, September 16-18

Parallel Multiple-Symbol Variable-Length Decoding

Jari Nikara[†], Stamatis Vassiliadis[‡], Jarmo Takala[†], Mihai Sima[‡], and Petri Liuha[§]

[†]Institute of Digital and Computer Systems, Tampere University of Technology, Tampere, Finland
[‡]Computer Engineering Lab., Dept. of Electrical Engineering, TU Delft, Delft, The Netherlands
[§]Nokia Research Center, Tampere, Finland
E-mail: jari.nikara@tut.fi

Abstract

In this paper, a parallel Variable-Length Decoding (VLD) scheme is introduced. The scheme is capable of decoding all the codewords in an N-bit buffer whose accumulated codelength is at most N. The proposed method partially breaks the recursive dependency related to the MPEG-2 VLD. All possible codewords in the buffer are detected in parallel and the sum of the codelengths is provided to the external shifter aligning the variable-length coded input stream for a new decoding cycle. Two length detection mechanisms are proposed: the first approach determines the length in a parallel/serial fashion and the second using a new device denoted as MultiplexedAdd. In order to prove feasibility and determine the limiting factors of our proposal, the parallel/serial codeword detector with 32-bit input has been described in behavioral non-optimized VHDL and mapped onto Altera's ACEX EP1K100 FPGA. The implemented prototype exhibits a latency of 110 ns and uses 32% of the logic cells of the device. When applied to MPEG-2 standard benchmark scenes, on average 3.5 symbols are decoded per cycle.

1. Introduction

The Variable-Length Coding (VLC) is used as a mean of compression of image and video sequences. As its name indicates, the codewords are of variable-length. Furthermore, in the MPEG-2 standard, there is no boundary information for detecting the end or beginning of the codeword. The above substantially complicates the design and performance of Variable-Length Decoding (VLD) hardware realizations.

A traditional way to manage the complexity is to decode one symbol at time. There are two hardware approaches: the serial tree-based processing, resulting in *constant input / variable output rates* decoding [3, 6, 8] and the bit-parallel approach with *variable input / constant output rates* [7]. When considering multiple-symbol decoding schemes, the main design issues are the breaking of the data dependencies between codewords, which excludes the serial processing, and the management of the increasing hardware and control complexity, especially with large code tables and long codewords. According to the properties of VLC, most probably a block of bits in the input stream contains more than one codeword. This fact has been exploited in a *variable input / variable output rate* multiple-symbol decoding schemes for short codewords operating on the longest codeword length buffer proposed in [1, 4]. The alternative way to manage complexity is to keep the output rate constant [12]. In the current multiple-symbol approaches, the performance is limited due to the fact that the long arbitrary length input buffers are not exploited. Two possible implementations are available where either only short codewords are decoded concurrently or the number of symbols is limited.

This paper describes a new *variable input / variable output* VLD with the following main contributions:

- We propose a multiple-symbol parallel decoding scheme, which decodes all the complete codewords stored into the input buffer of arbitrary length. All the possible codewords in the buffer are detected in parallel and the sum of the codeword lengths is provided to an external shifter aligning the variable-length coded input stream for the next decoding cycle.
- We propose two mechanisms with the intend to provide short critical paths. The first mechanism determines the length in a parallel/serial fashion and the second introduces the MultiplexedAdd unit, which fuses the critical path and almost reduces by half the critical path in terms of logic gates.
- We provide a prototype based on Altera's ACEX EP1K100 FPGA intended to show the limiting features of our approach. We show that a naive implementation requires 32% of the FPGA logic cells, has 110 ns cycle time it is capable of detecting in average 3.5 symbols of the 4.7 potential symbols detected out of a 32-bit buffer.

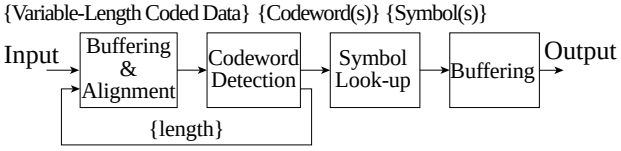

Figure 1. Generalized VLD scheme.

The remaining of the discussion is organized as follows. Related work is outlined in Section 2. In Section 3, the proposed decoding scheme is introduced and the theoretical performance is estimated. Decoder design and experimental results are discussed in Section 4. Finally, the conclusions are presented with a glance to future work in Section 5.

2. Related Work

Hardwired VLD decoders extract the codeword and its length and aligns the variable-length coded input stream for the next decoding iteration as illustrated in Fig. 1. Consequently, depending on codeword and prespecified code values, i.e., code table, symbols are determined. Depending on the decoding technique, input code, output symbols, or both are buffered. Existing VLD decoders can be classified in three approaches as follows:

Approach 1: The serial architectures, also referred to as tree-based architectures, decode data sequentially, bit-by-bit [8] or in clusters of several bits [3]. The algorithm used is the inverse interpretation of building the Huffman tree; coded input stream is compared to binary tree starting at the root of the tree. The comparison is performed with a *constant input rate*, one bit per cycle, until the entire codeword is detected in corresponding leaf node resulting in a *variable output rate*. Short decoding time is achieved only with short codewords. However, under hard real-time constraints, the required output rate should be fulfilled also with long codewords, thus the performance is defined by the latency of the long codeword processing. Furthermore, the serial processing is not applicable for multiple-symbol decoding due to recursive dependencies between codewords.

Approach 2: For a *constant output rate*, the number of bits to be decoded at a time should be equal to the longest codelength resulting in bit-parallel processing, which guarantees that at least one codeword is detected. Traditionally, codeword has been detected with pattern matching based on logical functions [7]. The alignment of input stream for the next cycle is performed according to the codelength. Advances are achieved by clustering bit patterns and utilizing tree-based pattern matching [2]. Moreover, designs can be pipelined into stages of codelength determination and finding the corresponding symbol since length information is sufficient to extract codeword [9]. Furthermore, the traditional pattern matching has been replaced with arithmetic operations utilizing the properties of codeword table, e.g., leading characters and numerical properties [10, 11, 13].

Approach 3: According to the properties of VLC, most probably a block of bits in the input stream contains more than one codeword. This fact has been exploited in a *variable input/output rate* multiple-symbol decoding scheme for short codewords proposed in [1, 4]. The exponentially increasing control and hardware complexity sets constraints to implementations, especially, when large code tables are used. Hence, the number of bits to be decoded is limited to the longest codelength [1] or alternatively the number of outputs is limited [4]. The increasing complexity can also be managed by using symbol parallel decoding while keeping the output rate constant [12].

In this paper, we propose a multiple-symbol decoding scheme, which is parallel (different from [3, 6, 8]). It decodes multiple symbols (different from [2, 3, 6, 8, 9, 10, 13]) and exploits arbitrary codelength buffers and variable output rate (different from [1]- [4], [6]- [13]). Finally, we propose a specific hardware mechanism, which improves the cycle time of the decoder.

3. Decoding Scheme

The main challenge in the parallel symbol detection in VLD is to break the recursive dependencies between the codewords or at least to minimize its effects to the throughput. The proposed approach is to decode all the codewords stored into the codeword buffer simultaneously.

To achieve our goals, we first determine how many variable-length codewords can exist in the codeword buffer at a time. To this purpose we define the codelengths of a code table with the aid of a set $S_L = \{l_1, \ldots, l_n\}$ where l_1 and l_n denote the minimum and maximum length of codewords, respectively. Consequently, the maximum number of codewords in an N-bit buffer is $K_{max} = \lfloor N/l_1 \rfloor$, $N \geq l_n$. Let us denote the variable-length codewords in the buffer by W_i where $i = 0, 1, \ldots, (K_{max} - 1)$ and the length of codeword W_i by L_i. Moreover, let us define an index j_i, $0 \leq j_i \leq (N - 1)$, which indicates the first bit of the codeword W_i in the N-bit codeword buffer.

For ease of comprehension and without losing generality, we may assume that the first codeword W_0 is always located in the beginning of the buffer thus $j_0 = 0$. The second codeword W_1 is located immediately after the first codeword, thus the index indicating the start of the second codeword is the length of the first codeword, i.e., $j_1 = L_0$. This implies that the start index of the codeword W_i is the sum of the lengths of the previous codewords, i.e., $j_i = \sum_{k=0}^{i-1} L_k$. The lengths of the codewords in the buffer are not known in advance. In order to avoid the recursive dependencies, a parallel search is needed for the codewords from "arbitrary"

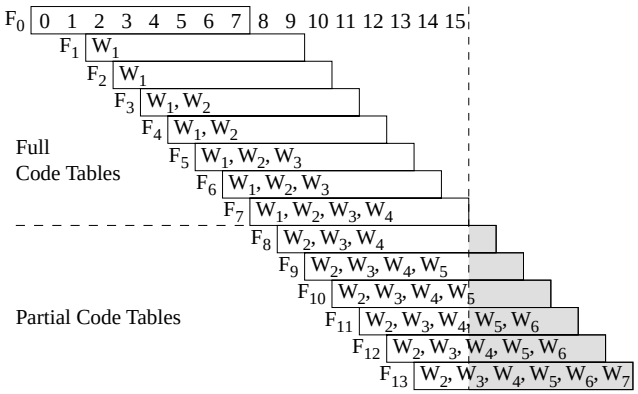

Figure 2. Principle of codeword detection.

locations in the buffer.

In general, the set of all the candidate indices can be defined as $p = \{0, l_1, (l_1 + 1), (l_1 + 2), \ldots, (N - l_1)\}$, which implies that there are $(N - 2(l_1 - 1))$ locations in the N-bit buffer where a codeword can be located. Since the maximum length of the codeword is known, i.e., l_n, we extract l_n-bit fields from all the possible locations defined by set p and apply pattern matching to detect a valid codeword in each field. However, the codeword detection can be performed only if all the bits of the codeword are available. Therefore, in fields starting at the last $(l_n - 1)$ indices, the pattern matching is easier; fields starting at index $N - K, K < l_n$, only codewords of lengths up to K bits need to be searched after.

The previous procedure will detect a redundant number of codewords. The reason is that a shorter codeword can be found from a valid codeword when the bit field is extracted in the middle of the valid codeword. Therefore, each search process returns only the length of the detected codeword. With the aid of the lengths, we may define the indices of the valid codewords in the buffer; the length of the first codeword defines the index to the second codeword. The lengths of the first and second codeword define the index to the third codeword, etc.

An example of detecting the codewords in 16-bit buffer is illustrated in Fig. 2. Assuming a code table whose codelengths are defined by the set $S_L = \{2, 3, 4, 5, 6, 7, 8\}$, the maximum number of codewords is $K_{max} = 8$. In this case, 14 bit fields are extracted and all the codewords are matched into these fields as illustrated with the aid of boxes in Fig. 2. The first field, F_0 consist of the first valid codeword W_0. The second codeword is found in one of the seven fields F_1 - F_7. Similarly, the possible third codeword can be found from the fields F_3 to F_{13}. The possible codewords in the bit fields are included into corresponding boxes in Fig. 2. The lengths of bit fields from F_8 to F_{13} are shorter than the others, since they are in the end of the buffer and the number of available bits in the buffer is less l_n. This is indicated by the grey area of the boxes in Fig. 2.

Table 1. Properties of benchmarks.

Benchmark	b	S	B	b/B	S/B
bat_327_334	1 506 680	266 485	38 940	38.7	6.8
popplen	153 265	28 069	4 139	37.0	6.8
sarnoff	169 567	36 408	8 447	20.1	4.3
tennis	989 235	137 756	25 524	38.8	5.3
t1cheer	255 433	51 680	9 432	27.1	5.6
Total	3 074 180	520 398	86 482	35.6	6.0

b:bits. S:symbols. B:block. b/B:bits per block. S/B:symbols per block.

In order to complete variable-length decoding, the symbols corresponding codewords should be searched from a code table. Since the codeword boundary information is obtained from the codeword detection described previously, the recursive dependencies between codewords are removed. In other words, the codewords can be extracted from input stream and look-up process can be performed independently. Briefly, the described variable-length decoding scheme can be outlined as follows:

- The maximum number of codewords the N-bit codeword buffer can hold K_{max} is determined
- $(N - 2(l_1 - 1))$ bit fields of size l_n bits are extracted from the buffer. The bit fields are extracted from locations $\{0, l_1, l_1 + 1, l_1 + 2, \ldots, N - l_1\}$
- Codewords are detected from each bit field such that the possible codeword starts from the beginning of the field. If a codeword is detected, the length of the codeword is returned.
- The valid codewords in the buffer are found according to indices, which are obtained by computing the sum of the lengths of the previous valid codewords.
- Symbols corresponding the valid codewords are found with the aid of table look-up process where parallelism can be increased.

The highest utilization rate of the buffer is achieved if all the complete codewords are detected in a single cycle and the codeword buffer can be updated at each cycle. However, in practice, the buffer may contain a partial codeword, which should be kept in the buffer and processed at the next cycle when the remaining bits are fetched into the buffer. In order to estimate the upper bound for the throughput, the scheme is applied to MPEG-2 benchmark scenes coded according to code table B.14 in [5]. The properties of benchmarks are summarized into Table 1, and the proportion of buffer size to throughput is illustrated in Fig. 3.

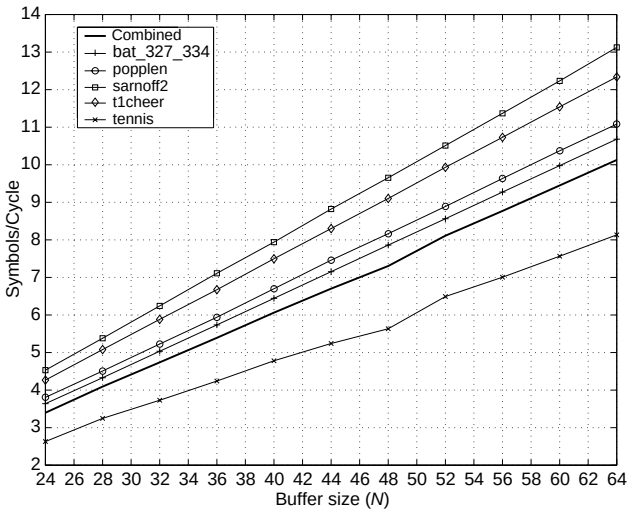

Figure 3. Register size vs. throughput.

4. Variable-Length Decoder and Experimental Results

Since the proposed variable length decoder results in a *variable input/variable output rate* system, the buffering resources are needed in the input as well as in the output. The target for our design is an embedded system with external buffering and shifting resources. In the presentation, only the kernel design of the VLD consisting codeword detection and symbol lookup is considered.

General Organization: The decoder design is started by considering the parallel detection of all the codewords in the input buffer. This is realized with $(N - 2(l_1 - 1))$ Codeword Detectors (CD) as illustrated in Fig. 4. With this arrangement, the $(N - l_n - l_1 + 2)$ leftmost CDs obtain an l_n-bit field from the input buffer but from different bit locations while for the remaining $(l_n - l_1)$ CDs, it is sufficient to detect only shorter codewords. All the CDs detect codewords simultaneously and return the length of the detected

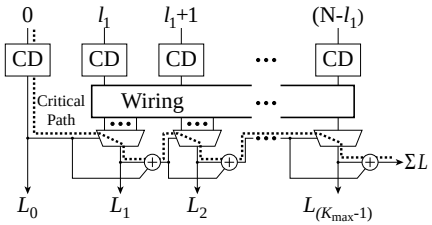

Figure 4. Organization of parallel/serial codeword detection.

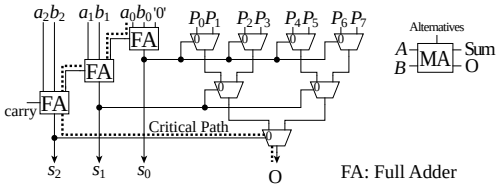

Figure 5. Schematic of MultiplexedAdd.

codeword. In order to select the valid codelengths, i.e., L_i, from all the lengths of detected codewords, the stage of cascaded multiplexers is employed as depicted in Fig. 4. Each multiplexer should have inputs from all the CDs whose bit fields are in locations il_1–il_n in the input buffer. Since the first codelength L_0 is always obtained from the first CD, it controls the first multiplexer selecting the second valid codelength L_1. Moreover, the output of the first output can be used to provide the decoding status, i.e., if the codelength is zero, the decoding is completed or an error is detected. The other multiplexers are controlled by the sum of the previous codelengths. Hence, the computation of the sum of the codelengths creates the critical path, which is shown with a dashed line in Fig. 4.

The codewords can be extracted from the input buffer according to the length information and decoded independently. Moreover, the decoding can be parallelized when the recursive dependencies are removed in the codeword detection. The performance bottleneck will be the codeword detection, which should be one cycle operation in order to obtain the align information, i.e., the sum of codelengths in the buffer to shifter. In order to minimize the latency of critical paths in the codeword detection, the **MultiplexedAdd (MA)** component is introduced. The MA computes the sum of two inputs and performs multiplexing in parallel with the addition. To clarify consider to following. Let us assume two three-bit numbers A and B whose sum S controls the selection of the output O from possibilities $P_0 - P_7$. Consequently, the output can be defined as

$$\begin{aligned} O =& P_0\bar{s_2}\bar{s_1}\bar{s_0} + P_1\bar{s_2}\bar{s_1}s_0 + P_2\bar{s_2}s_1\bar{s_0} + P_3\bar{s_2}s_1s_0 + \\ & P_4s_2\bar{s_1}\bar{s_0} + P_5s_2\bar{s_1}s_0 + P_6s_2s_1\bar{s_0} + P_7s_2s_1s_0, \quad (1) \end{aligned}$$

which can be further decomposed as

$$\begin{aligned} O =& \left(P_0\bar{s_1}\bar{s_0} + P_1\bar{s_1}s_0 + P_2s_1\bar{s_0} + P_3s_1s_0\right)\bar{s_2} + \\ & \left(P_4\bar{s_1}\bar{s_0} + P_5\bar{s_1}s_0 + P_6s_1\bar{s_0} + P_7s_1s_0\right)s_2 \\ =& \left[\left(P_0\bar{s_0} + P_1s_0\right)\bar{s_1} + \left(P_2\bar{s_0} + P_3s_0\right)s_1\right]\bar{s_2} + \\ & \left[\left(P_4\bar{s_0} + P_5s_0\right)\bar{s_1} + \left(P_6\bar{s_0} + P_7s_0\right)s_1\right]s_2. \quad (2) \end{aligned}$$

The corresponding logic design is depicted in Fig. 5. With the aid of this unit, the sum of current codelength L_i, and previous codelengths, i.e., $\sum_{k=0}^{i-1} L_k$, can be computed and

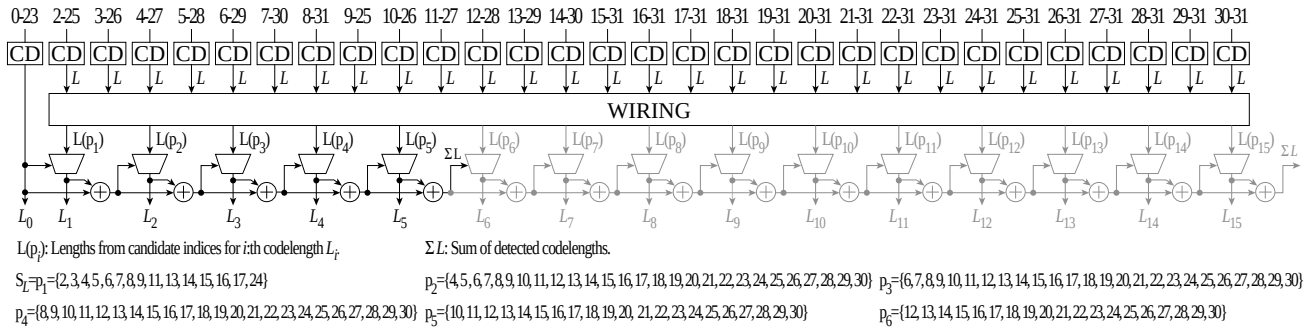

Figure 6. Organization of FPGA codeword detection.

the next codelength L_{i+1} can be selected at the same time. Using MAs, the latency between two codelengths is reduced from the latency of a $\log(N)$-bit adder and complex multiplexers to the latency of $\log(N)$ full adders and a 2-1 multiplexer as illustrated in Fig. 5. In terms of logical stages using 3-4 And-Or (AO) and 2-2 AOs and inverters, the stages can be reduced from $2\log(N)$ stages to $\log(N)+1$ stages.

Demonstrator (codeword detection) and performance analysis: In order to estimate the worst latency of the proposed decoding scheme, the codeword detector returning codelength was described in behavioral VHDL and realized on Altera's ACEX EP1K100 FPGA. In order to establish the worst case scenario for our scheme we implemented the organization reported in Fig. 4. Additionally, the realization does not contain any FPGA specific optimizations. Continuous MPEG-2 data coded according to codeword table B.14 in [5] has been chosen as input for the implementation.

The design specifications are determined according to analyzed statistics of source data in Table 1. In MPEG-2 data, the DC coefficient, i.e. the first codeword in a block, is interpreted differently from the AC coefficients. In our implementation, this dependency between the blocks is simplified by decoding the DC coefficient only in the first CD. The drawback is that when end-of-block codeword is detected, the buffer is updated although other codewords may still exist in the buffer. In other words, only one block can be processed at a time. Therefore, the input buffer has been specified to be 32 bits while the number of bits in a block varies from 20 to 39. Since $l_1 = 2$ and $l_n = 24$, the resulting structure consist of 30 CDs as depicted in Fig. 6. The eight leftmost CDs can detect the all codewords in the code table, thus they have 24-bit inputs aligned to different buffer bit locations shown above the CDs in Fig. 6. The next seven CDs have 17-bit inputs and the input width of all the other CDs decreases according to codelengths until the last CD detects only 2-bit codewords. All the CDs return codelengths in parallel. The codelength set S_L for B.14 is depicted in Fig. 6. The buffer may contain at most 16 codewords, which determines the number of outputs. The largest group of codelength candidates is for the third output L_2, which consist of lengths from 28 CDs. The codelength candidates for the six outputs are illustrated with the aid of set p_i defining the starting locations of CDs in Fig. 6.

We have experimented with two designs. The first detects all symbols from the buffer and the other detects at maximum six symbols from the buffer. The cycle time of **the first design** (Fig. 6 including all blocks) is defined by the signal delay through a CD with 24-bit input, one 15-1 multiplexer and 15 five-bit adders. The synthesis of behavioral VHDL results in a latency of 250 ns, which proves the feasibility and shows the limit of the approach rather than its potential. The experimental results are summarized into Table 2. Column "Scheme" contains the upper limits for the performance scheme with a 32-bit buffer. The figures are obtained by assuming that data is processed without making any difference to the DC and AC coefficients and all the codewords in the input buffer are detected concurrently. The required cycles and achieved throughput for the demonstrator are depicted in column "FPGA-32/16" in Table 2. On average, 3.6 codewords per cycle are detected, which differs from theoretical values due to avoiding block dependencies.

Table 2. Experimental results.

	Scheme		FPGA-32/16		FPGA-32/6	
Benchmark	C	W/C	C	W/C	C	W/C
bat_327_334	52 928	5.0	70 018	3.8	71 356	3.7
popplen	5 369	5.2	7 141	3.9	7 345	3.8
sarnoff	5 834	6.2	10 452	3.5	10 725	3.4
tennis	36 920	3.7	44 342	3.1	44 733	3.1
t1cheer	8 784	5.9	13 301	3.9	13 807	3.7
Total	109 835	4.7	145 254	3.6	147 966	3.5

Scheme: scheme (32-b input, 16 outputs). FPGA-X/Y: demonstrator (X-b input, Y outputs). C: cycles. W/C: codewords per cycle.

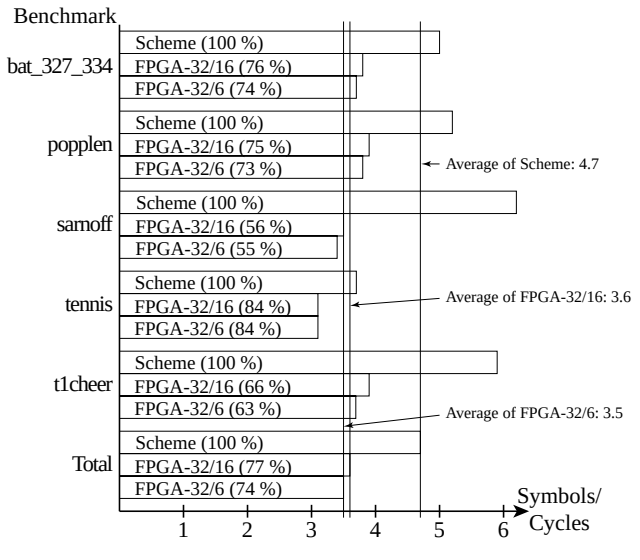

Figure 7. Throughput comparison.

Since we consider one block at a time and because the average symbols per block is six, the number of outputs is reduced from 16 to six in **the second design** (Fig. 6 only dark lined blocks). This was justified by the fact that the average number of symbols per block in our benchmarks is about six. Consequently, the latency of the multiplexer chain is shortened. However, by maintaining the size of the input buffer, the probability to have six symbols at time was increased. The design is depicted in Fig. 6 where the removed parts are drawn with lighter lines. The design possesses the latency of 110 ns. Extra cycles are required if block contains more than six symbols. The cost of the simplification in terms of total number of cycles is presented in column "FPGA-32/6" in Table 1 whereas the differences between the benchmarks are illustrated in Fig. 7. It is noted that the performance of the two designs FPGA-32/16 and FPGA-32/6 in terms of detected symbols per cycle is very close while the cycle time of FPGA-32/16 is more than double of the FPGA-32/6.

5. Conclusions

In this paper, a parallel multiple-symbol decoding scheme for variable-length codes has been proposed. The proposed scheme is applied to MPEG-2 benchmark scenes for estimating the maximum performance achievable. It has been shown that the throughput rate is proportional to the size of input buffer and for 32-bit buffer, the average throughput is 4.7 symbols per cycle. Two schemes have been proposed for providing a VLD and a naive codeword detector has been described in VHDL and mapped onto Altera's ACEX EP1K100 FPGA. The evaluated results indicate that 3.5 symbols per cycle out of the 4.7 average symbols present in the 32-bit buffer can be detected per cycle. The critical path of 110 ns, proves the feasibility and is a limiting factor of the approach rather than its potential. In the future, we intend to design a more structured fast codeword detection. In addition, parallel symbol search is studied for implementing the variable-length decoder in its entirety.

References

[1] S.-F. Chang and D. G. Messerschmitt. Designing high-throughput VLC decoder. Part I - Concurrent VLSI architectures. *IEEE Trans. Circuits Syst. Video Technol.*, 2(2):187–196, June 1992.

[2] S. B. Choi and M. H. Lee. High speed pattern matching for a fast Huffman decoder. *IEEE Trans. Consumer Electron.*, 41(1):97–103, Feb. 1995.

[3] R. Hashemian. Design and hardware implementation of a memory efficient Huffman decoding. *IEEE Trans. Consumer Electron.*, 40(3):345–352, Aug. 1994.

[4] C.-T. Hsieh and S. P. Kim. A concurrent memory-efficient VLC decoder for MPEG applications. *IEEE Trans. Consumer Electron.*, 42(3):439–446, Aug. 1996.

[5] International Telecommunication Union. Information technology – Generic coding of moving pictures and associated audio information: Video. ITU-T Recommendation H.262, Feb. 2000.

[6] Y.-S. Lee, B.-J. Shieh, and C.-Y. Lee. A generalized prediction method for modified memory-based high throughput VLC decoder design. *IEEE Trans. Circuits Syst. II*, 46(6):742–754, June 1999.

[7] S. M. Lei and M. T. Sun. An entropy coding system for digital HDTV applications. *IEEE Trans. Circuits Syst. Video Technol.*, 1(1):147–155, Mar. 1991.

[8] A. Mukherjee, N. Rangnathan, and M. Bassiouni. Efficient VLSI designs for data transformation of tree-based codes. *IEEE Trans. Circuits Syst.*, 38(2):306–314, Mar. 1991.

[9] M. K. Rudberg and L. Wanhammar. New approaches to high speed Huffman decoding. In *Proc. IEEE Int. Symp. Circuits Syst.*, volume 2, pages 149–152, Atlanta, USA, May 1996.

[10] B.-J. Shieh, Y.-S. Lee, and C.-Y. Lee. A new approach of group-based VLC codec system with full table programmability. *IEEE Trans. Circuits Syst. Video Technol.*, 11(2):210–221, Feb. 2001.

[11] M. Sima, S. Cotofana, S. Vassiliadis, J. T. J. van Eijndhoven, and K. Visser. MPEG macroblock parsing and pel reconstruction on an FPGA-augmented TriMedia processor. In *Proc. IEEE Int. Conf. Comput. Design*, pages 425–430, Austin, Texas, USA, Sep. 24–26 2001.

[12] M. Sima, S. Cotofana, S. Vassiliadis, J. T. J. van Eijndhoven, and K. Visser. MPEG-compliant entropy decoding on FPGA-augmented TriMedia/CPU64. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa Valley, CA, USA, Apr. 21–24 2002.

[13] B. W. Y. Wei and T. H. Meng. A parallel decoder of programmable Huffman codes. *IEEE Trans. Circuits Syst. Video Technol.*, 5(2):175–178, Apr. 1995.