Multiple-Symbol Parallel Decoding for Variable Length Codes

Jari Nikara, *Student Member, IEEE*, Stamatis Vassiliadis, *Fellow, IEEE*, Jarmo Takala, *Senior Member, IEEE*, and Petri Liuha

***Abstract*—In this paper, a multiple-symbol parallel variable length decoding (VLD) scheme is introduced. The scheme is capable of decoding all the codewords in an N-bit block of encoded input data stream. The proposed method partially breaks the recursive dependency related to the VLD. First, all possible codewords in the block are detected in parallel and lengths are returned. The procedure results redundant number of codeword lengths from which incorrect values are removed by recursive selection. Next, the index for each symbol corresponding the detected codeword is generated from the length determining the page and the partial codeword defining the offset in symbol table. The symbol lookup can be performed independently from symbol table. Finally, the sum of the valid codeword lengths is provided to an external shifter aligning the encoded input stream for a new decoding cycle. In order to prove feasibility and determine the limiting factors of our proposal, the variable length decoder has been implemented on an FPGA technology. When applied to MPEG-2 standard benchmark scenes, on average 4.8 codewords are decoded per cycle resulting in the throughput of 106 million symbols per second.**

***Index Terms*—Critical-path, design, gate-array, image-processing, reconfigurable-systems, video-processing.**

I. Introduction

THE ultimate purpose of compression is to represent a set of symbols with minimum number of bits. This is achieved by representing frequently occurring symbols with shorter codewords. Such a coding method results in variable codeword lengths hence the name variable length coding (VLC). The theoretical lower bound on the average number of bits required to represent a symbol in the given set is defined by entropy [1]. In order to reach entropy, noninteger codeword lengths are needed. Suboptimal compression can be obtained with integer codeword lengths and a coding method providing the shortest integer length codewords is Huffman coding [2].

The inverse process for VLC is variable length decoding (VLD) where the codeword length is detected from a block of the variable length coded input stream and this codeword is used to determine the actual symbol with the aid of predefined codeword values, i.e., codeword table. The input stream is then aligned for the next decoding iteration as illustrated in Fig. 1. In general, there is no explicit boundary information for detecting the end or beginning of the codeword in the coded data stream. Therefore, the length of the current codeword should be known before the next codeword can be decoded. This feature complicates the decoder design substantially and limits the performance.

A traditional VLD method is to decode one symbol at time in symbol-serial fashion. Two principal approaches exist: the bit-serial tree-based processing resulting in *constant input/variable output rates* decoding [3]–[5] and the bit-parallel approach with *variable input/constant output rates* [6]. In multiple-symbol decoding or symbol-parallel schemes, the major design issue is to break the data dependencies between codewords. Another issue is the management of the increasing hardware and control complexity, especially when large codeword tables and long codewords are used.

Often 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 [7], [8], which operate on a buffer whose size is equal to the longest codeword. An alternative method is to keep the output rate constant [9], [10]. However, in the current multiple-symbol approaches, the performance is limited due to the fact that the arbitrary length input buffers are not exploited. In the previous methods, either only short codewords are decoded concurrently or the number of symbols is limited.

In this paper, a novel multiple-symbol parallel VLD scheme is proposed and applied in MPEG-2 VLD. The work is based on the work reported earlier in [11]. The main contributions of this paper are the following.

1) *Multiple-symbol parallel decoding scheme*, which decodes all the complete codewords in an arbitrary length block of input data.
2) *Multiplexed add unit*, which reduces the number of logic levels in the critical path of the parallel/serial codeword detection.
3) *MPEG-2 decoder demonstration on a field-programmable gate array (FPGA)*, which proves the feasibility and illustrates the limitations of the approach. It is shown that a technology independent hardware description on the FPGA technology results in a cycle time of 45 ns. On average, the demonstration can detect

Manuscript received January 13, 2003; revised July 3, 2003. This work was supported in part by the Academy of Finland, under Project 50554, in part by the Graduate School of Electronics, Telecommunications, and Automation (GETA), in part by the Jenny and Antti Wihuri Foundation, in part by the Ulla, Tuominen Foundation, and in part by the Foundation of Advancement of Technology.

J. Nikara is with Tampere University of Technology, 33101 Tampere, Finland and Delft University of Technology, 2600 Delft, The Netherlands (e-mail: jari.nikara@tut.fi).

S. Vassiliadis is with Delft University of Technology, 2600 GA Delft, The Netherlands (e-mail: s.vassiliadis@et.tudelft.nl).

J. Takala is with Tampere University of Technology, 33101 Tampere, Finland (e-mail: jarmo.takala@tut.fi).

P. Liuha is with Nokia Research Center, 33721 Tampere, Finland (e-mail: petri.liuha@nokia.com).

Digital Object Identifier 10.1109/TVLSI.2004.825840

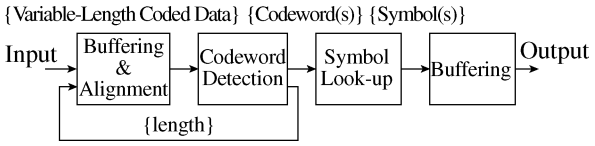

Fig. 1. Block diagram of generalized variable length decoding.

4.8 symbols of the 5.6 potential symbols when a 31-bit input buffer is used.

The remaining of the discussion is organized as follows. The previous work is outlined in Section II. In Section III, the proposed decoding scheme is introduced and the theoretical performance is estimated. Decoder design is described in Section IV and experimental results are discussed in Section V. Finally, the conclusions are presented with a glance to future work in Section VI.

II. Previous Work

Existing VLC decoders can be classified into three approaches as follows:

A. Serial Decoders

The serial architectures, also referred to as tree-based architectures, decode input data stream sequentially, bit-by-bit [3] or in clusters of several bits [4]. The used algorithm is the inverse interpretation of building the Huffman tree; coded input stream is compared to a 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 the corresponding leaf node. Due to the variable codeword lengths, the serial processing results 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 the recursive dependencies between the codewords.

B. Parallel Decoders

For a *constant output rate*, the number of bits to be decoded at a time should be equal to the longest codeword length resulting in bit-parallel processing, which guarantees that one codeword is detected at each cycle. Traditionally, codewords are detected with pattern matching based on logical functions [6]. The alignment of input stream for the next cycle is performed according to the codeword length. Advances are achieved by clustering bit patterns and utilizing tree-based pattern matching [12]. Moreover, designs can be pipelined into stages of codeword length determination and finding the corresponding symbol since the length information is sufficient to extract a codeword [13]. Furthermore, the traditional pattern matching has been replaced with arithmetic operations utilizing the properties of codeword table, e.g., leading characters and numerical properties [14]–[16].

C. Multiple-Symbol Decoders

According to the properties of the VLC, most probably a block of bits in the input stream contains more than one codeword. This fact has been exploited in *variable input/output rate* multiple-symbol decoding schemes for short codewords in [7], [8]. The exponentially increasing control and hardware complexity sets constraints to implementations, especially, when large codeword tables are used. Hence, the number of bits to be decoded is limited to the longest codeword length [7] or alternatively the number of outputs is limited [8]. The increasing complexity can also be managed by using symbol parallel decoding while keeping the output rate constant [9], [10].

In this paper, we propose a multiple-symbol variable length decoding scheme with the following properties; the scheme a) is parallel, b) decodes multiple symbols, and c) exploits arbitrary codelength buffers and variable output rate. The property a) is different from serial decoders, property b) is different from parallel decoders, and property c) is different from existing parallel and multiple-symbol decoders. Finally, we propose a specific hardware mechanism, which shortens the critical path of the decoder implementation.

III. Decoding Scheme

The main challenge in the multiple-symbol parallel VLD is to break the recursive dependencies between the codewords or at least to minimize their effects to the throughput. The proposed approach is to decode all the codewords in a block of input data stream simultaneously. In this section, a VLD scheme is introduced and illustrated with an example. A general hardware organization is proposed with an illustration and its performance is discussed.

A. Algorithm

Let us assume K symbols and the corresponding codewords are collected into a codeword table $C_k = (c_0^k, \ldots, c_{m_k-1}^k) | c_i^k \in \{0,1\}, k = 0, \ldots, K-1$. All the different codeword lengths in the codeword table can be combined into a set L defined as $L = \bigcup_{k=0}^{K-1} \{m_k\}$. Let the minimum and maximum codeword lengths be denoted by l_1 and l_n, respectively. In addition, the maximum number of codewords with equal length is denoted by $d_{\max}$. We use a group-based approach for storing the symbols into a symbol table; the symbols are grouped according to the length of the corresponding codeword and each group is stored into one page in the table. The size of the page is defined by $d_{\max}$. In such an arrangement, the page where the symbol S_k is stored is determined by the length of its codeword, m_k. The symbols within a page are arranged in such a way that the offset within the page is determined by the Least Significant Bits (LSB) of the codeword, $(c_{\log_2 d_{\max}-1}^k, \ldots, c_0^k)$.

The input data stream for the decoding process is an encoded binary vector X, i.e., $X = (x_0, x_1, x_2, \ldots)$, $x_i \in \{0,1\}$. An N-bit sliding window B is used to extract bits from the input stream as $B = (b_0, b_1, \ldots, b_{N-1}), b_i = x_{idx+i}, i = 0, \ldots, N-1$ where idx is the index to the first undecoded bit in the input stream X. Throughout the discussion, the sliding window B is assumed to be greater than the longest codeword, i.e., $N \geq l_n$.

We start the derivation of the algorithm by determining the maximum number of variable length codewords, M, in an N-bit sliding window B as

$$M = \left\lfloor \frac{N}{l_1} \right\rfloor. \tag{1}$$

Let us denote the variable length codewords in the window by W_i where $i = 0, 1, \ldots, (M-1)$ and the length of codeword W_i by L_i. Moreover, let an index j_i, $0 \leq j_i \leq (N-1)$, define a location where the codeword W_i starts, i.e., $W_i = (b_{j_i}, \ldots, b_{j_i+L_i-1})$.

Without losing generality, we may assume that the first codeword W_0 is always located at the beginning of the window, thus $j_0 = 0$. The second codeword W_1 is located immediately after the first L_0-bit codeword and, therefore, W_1 can be found starting from the index $j_1 = L_0$. This implies that the start index of the codeword W_i in B is the sum of the previous codeword lengths, i.e.,

$$j_i = \sum_{k=0}^{i-1} L_k. \tag{2}$$

However, the lengths of the codewords are not known in advance.

In order to avoid the recursive dependencies, a parallel search is needed to find codewords from "arbitrary" positions in the window. In general, all the candidates for indices j_i for the codeword W_i can be represented with the aid of set J_i defined recursively as

$$J_0 = 0; J_i = \{j_i | j_i = q + l, \forall q \in J_{i-1}, \forall l \in L\} \tag{3}$$

which implies that a codeword can lie in any location in the window defined by a set J defined as

$$J = \bigcup_{k=0}^{M-1} J_k. \tag{4}$$

Since the maximum length of the codeword, l_n, is known, we need to extract at most l_n-bit fields from the window B starting from all the locations defined by set J. In each bit field, the possible codeword is searched after by matching the bit field with all the possible codewords. When a match is found, the length of the codeword at position i in the window B, p_i, is returned as

$$p_i = \begin{cases} m_k, & \text{if } (b_i, b_{i+1}, \ldots, b_{i+m_k-1}) = (c_0^k, c_1^k, \ldots c_{m_k-1}^k) \\ 0, & \text{otherwise} \end{cases}$$

where $i \in J$, $k = 0, 1, \ldots, K-1$.

The start index, j_i, of the each valid codeword W_i in the window can be defined with the aid of the lengths of the detected codewords. Correspondingly, the length of W_i is $L_i = p_{j_i}$. The symbol lookup is performed from the symbol table according to index A_i, which is formed by concatenating the length of the codeword and its LSBs. By returning the sum of all the valid codeword lengths, the input stream can be aligned for the next decoding iteration by updating the sliding window index, $idx' = idx + j_M$. The described procedure is iterated until the entire input stream is decoded.

Decoding Example: Let us assume that a codeword table depicted in Fig. 2(a) is used, thus the set of codeword lengths is defined as $L = \{2, 3, 4, 5, 6, 7, 8\}$ and the maximum number of codewords in a 16-bit window B is $M = 8$. In principle, the proposed approach would result in a 5-bit index to symbol table. However, the size of the symbol table can be easily decreased by noting that four LSBs are sufficient for each individual index. The resulting symbol table consisting of seven pages of two symbols is illustrated in Fig. 2(b). In the example case, a codeword can lie in 14 bit fields starting at locations $J = \{0, 2, 3, 4, \ldots, 14\}$ as illustrated with the aid of boxes below the window in Fig. 2(c). The fields at the end of the window are shorter than the others since the number of available bits in the window is less than $l_n = 8$. All the fields are matched with all the codewords and the length and LSB of each detected codeword are returned. The detected codeword in the bit field is shown inside the corresponding box in Fig. 2(c). In the example case, the lengths of the codewords at positions seven and eight in the window B are zero, which implies that the codewords were not detected. The fields containing a valid codeword are determined recursively using start indices j_i defined in (2). The first valid codeword W_0 is found from the first bit field at the beginning of the window, i.e., the first start index is $j_0 = 0$. The second codeword W_1 can be found in one of the seven fields starting at locations $J_1 = \{2, 3, 4, 5, 6, 7, 8\}$. Since the length of W_0 is $L_0 = 5$, the start index of W_1 is $j_1 = 5$. In Fig. 2(c), the detected valid codewords are marked with grey color. Index A_i for the symbol lookup is formed by concatenating the length and the LSB of the valid codeword. E.g., the length of W_1 is $L_1 = 4$ and the LSB of the W_1 is 0 and, therefore, index A_1 is 1000 and **D** is fetched from the symbol table.

B. General Organization

The previously discussed sliding window is realized as a N-bit codeword buffer and the codeword detection is performed by $|J|$ parallel codeword detector (CD) units. The input for each CD is a bit field of at most l_n bits, which is obtained from the codeword buffer locations in the set J defined in (4). All the CDs detect codewords simultaneously and return the length of the detected codeword. With this arrangement, the left-most CDs up to location $N - l_n$ search after all the codewords in the codeword table while, for the remaining CDs, it is sufficient to detect only shorter codewords.

In order to select the valid codeword lengths, i.e., L_i, from the lengths of all the detected codewords, a cascade of multiplexers is employed as depicted in Fig. 3(a). Each multiplexer should have inputs (lengths) from all the CDs in the locations specified by J_i defined in (3). The first codeword length L_0 obtained from the leftmost CD starting at bit location $j_0 = 0$ controls the first multiplexer selecting the second valid codeword length L_1. Moreover, the output of the leftmost CD can be used to provide the decoding status, i.e., if the codeword length is zero, either the decoding is completed or an error has encountered. The other multiplexers are controlled by the sum of the previous codeword lengths according to (2). Hence, the computation of the sum of the valid codeword lengths creates the critical path as shown in Fig. 3(a).

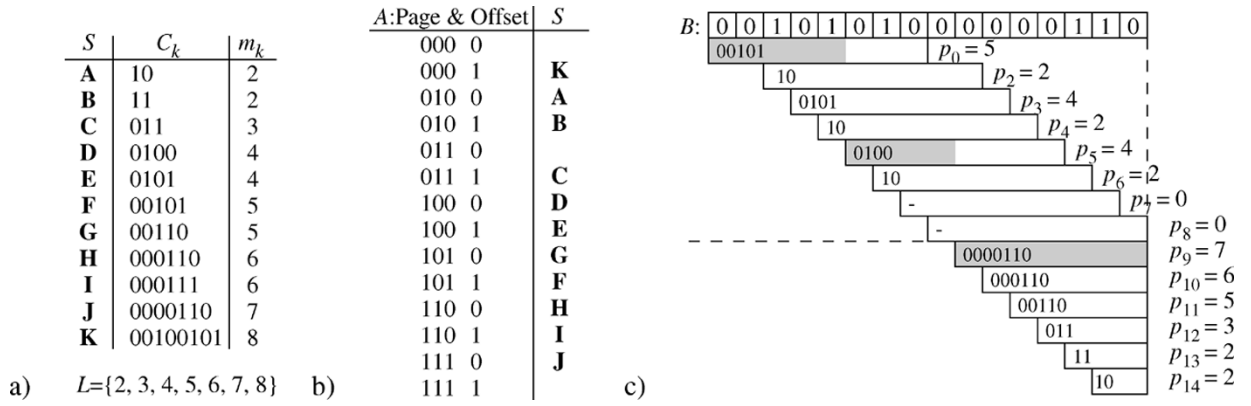

Fig. 2. Example of proposed variable length decoding. (a) Codeword table. (b) Symbol table. (c) Principle.

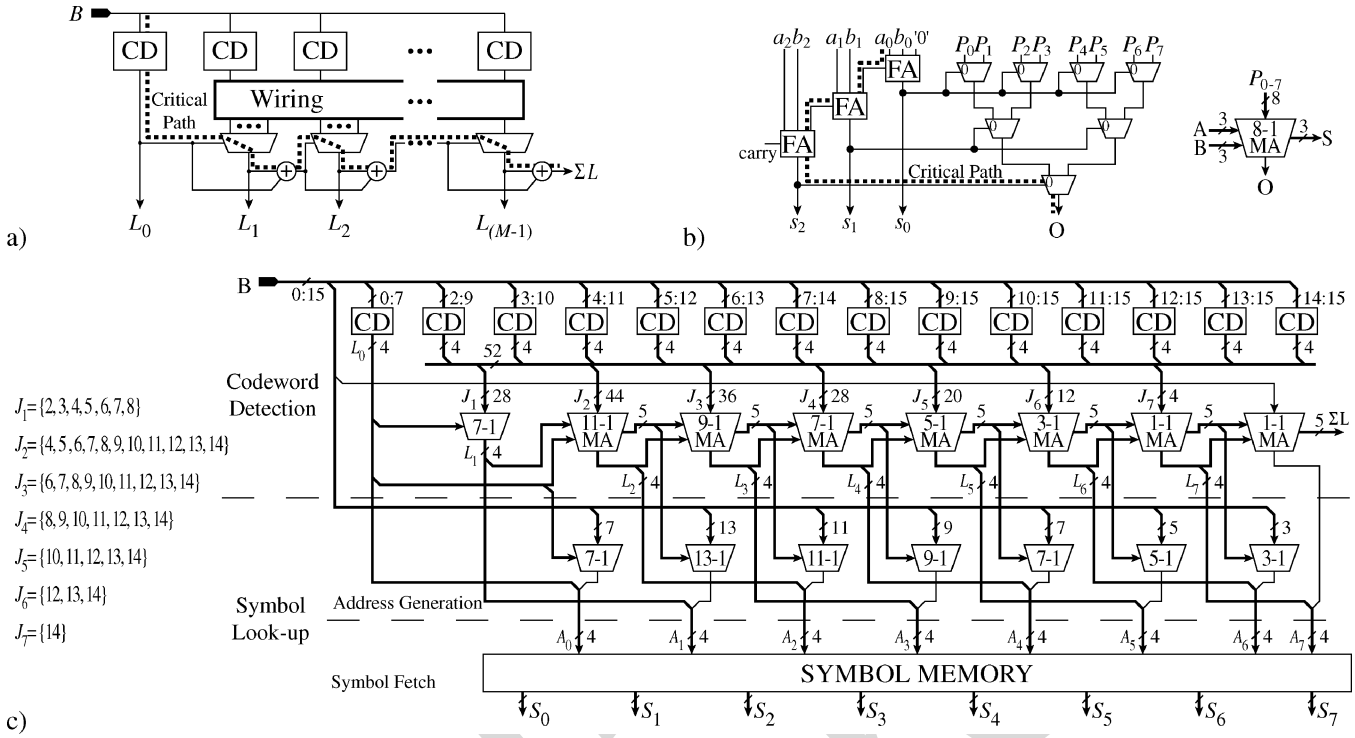

Fig. 3. Principal organization of scheme. (a) Generalized parallel/serial codeword detection. (b) 8–1 Multiplexed add. (c) Entire decoder.

For shortening the critical path, we introduce a multiplexed add (MA) unit shown in Fig. 3(b). In principle, the MA computes sum of two input operands, A and B, and the sum, S, is used to control a multiplexer selecting one of alternative inputs, P_i, to output O. In order to illustrate the operation of MA, let us assume two three-bit numbers $A = (a_2, a_1, a_0)$ and $B = (b_2, b_1, b_0)$. The sum denoted by $S = (s_2, s_1, s_0)$ controls the selection of the output O from inputs $P_0 - P_7$. Consequently, the output O can be expressed with the aid of sum of products as

$$\begin{aligned} O &= P_0\bar{s}_2\bar{s}_1\bar{s}_0 + P_1\bar{s}_2\bar{s}_1 s_0 + P_2\bar{s}_2 s_1\bar{s}_0 + P_3\bar{s}_2 s_1 s_0 \\ &\quad + P_4 s_2\bar{s}_1\bar{s}_0 + P_5 s_2\bar{s}_1 s_0 + P_6 s_2 s_1\bar{s}_0 + P_7 s_2 s_1 s_0 \\ &= (P_0\bar{s}_1\bar{s}_0 + P_1\bar{s}_1 s_0 + P_2 s_1\bar{s}_0 + P_3 s_1 s_0)\,\bar{s}_2 \\ &\quad + (P_4\bar{s}_1\bar{s}_0 + P_5\bar{s}_1 s_0 + P_6 s_1\bar{s}_0 + P_7 s_1 s_0)\,s_2 \\ &= [(P_0\bar{s}_0 + P_1 s_0)\,\bar{s}_1 + (P_2\bar{s}_0 + P_3 s_0)\,s_1]\,\bar{s}_2 \\ &\quad + [(P_4\bar{s}_0 + P_5 s_0)\,\bar{s}_1 + (P_6\bar{s}_0 + P_7 s_0)\,s_1]\,s_2. \end{aligned} \tag{5}$$

Closer examination of this decomposition reveals that each sum of products can be performed with the aid of 2–1 multiplexers. When MA is applied to the proposed VLD approach, the accumulated sum of the valid codeword lengths, i.e., the start index j_i, can be computed concurrently with the selection of current codeword length, L_i.

When the codeword length is known the symbol lookup is performed, i.e., a symbol corresponding to the valid codeword is fetched from the symbol table. The symbol table is mapped into a symbol memory as shown in Fig. 3(c). The symbol lookup can be decomposed into two phases: address generation and symbol fetch. Briefly, the address generation is used to form an address to symbol table, A_i, corresponding codeword W_i. The address A_i consists of page and offset where page forms most significant part of A_i. The page is the length of W_i, L_i, obtained from MA units as seen in Fig. 3(c). The offset consists of the

LSBs of the codeword, which can be determined according to start index of the next valid codeword j_{i+1}. If complex codeword tables, e.g., MPEG-2, are used, additional logic may be needed to form the page and offset. Finally, the symbol fetch is a trivial read memory operation. In order to support parallel symbol fetches, the symbol memory consists of separate parallel memory blocks, one for each decoder output, S_i.

Decoder Example: The principal organization of the entire VLD corresponding to the example illustrated in Fig. 2 is depicted in Fig. 3(c). In codeword detection, all the codewords in the 16-bit codeword buffer B are detected by 14 parallel CDs in defined locations. Each CD returns only the length of the detected codeword. The lengths of the valid codewords are selected by a 7–1 multiplexer and six cascaded 5-bit MA's. Each unit selecting L_i has lengths from the locations defined by set J_i. These locations are depicted on the left side of the input bus of the corresponding unit in Fig. 3(c). It should be noted that if no codeword matches the obtained bit field the MA returns zero, which is not, however, included into the number of alternatives denoted in the symbol of the MA. In the symbol lookup, the length of the valid codeword, L_i is used as a page. Since, the LSB of the codeword is enough to identify the codeword in Fig. 2(a), the LSB is extracted from the location $j_{i+1} - 1$ and used as an offset. Note that the extraction of the offsets resembles the selection of valid codewords: multiplexing controlled by accumulated length. Due to this similarity, the MA can be used not only to compute the final sum but to select the offset corresponding to the last codeword $W_{(M-1)}$. Finally, the symbol S_i can be fetched from the memory according to address A_i.

C. Critical Path

According to Fig. 1, the length of the detected codewords is used to align the data in the codeword buffer. This feedback path forms the critical path since the alignment and codeword detection should be performed in a single cycle. The critical path, according to Fig. 3(c), consists of a CD unit, $|J_1| - 1$ multiplexer, and a cascade of MA units. In order to approximate the critical path independent of technology, we use the interpretation from [17] where the delay is estimated with the aid of logical stages. A logical stage is assumed to be equivalent to 3–4 AND-OR (AO) and its delay is denoted by τ.

The number of AO stages in the CD unit is defined by the codeword table, which is application-specific. However, it is independent of N. Therefore, the delay of CD unit, t_D, is constant. The $|J_1| - 1$ multiplexer contains $\lceil \log_2(|J_1|) \rceil$ AO stages, thus the corresponding delay can be estimated as $t_M = \lceil \log_2(|J_1|) \rceil \tau < \lceil \log_2(N+1) \rceil \tau$. The codeword buffer may contain at most M codewords, thus the complete decoder contains $(M-1)$ cascaded MA units. The critical path through MA as seen in Fig. 3(b) consists of $\lceil \log_2(N+1) \rceil$ full adders and a 2–1 multiplexer, thus the delay of MA is $t_{MA} = (\lceil \log_2(N+1) \rceil + 1)\tau$. Therefore, the delay of the critical path of the decoder, t, is

$$\begin{aligned} t &\approx t_D + [\lceil \log_2(N+1) \rceil + (M-1)(\lceil \log_2(N+1) \rceil + 1)]\,\tau \\ &\approx t_D + [M(\lceil \log_2(N+1) \rceil + 1)]\,\tau. \end{aligned} \quad (6)$$

Although, the variable M according to the definition in (1) is dependent on N, we may interpret that M defines the number of outputs of the decoder, i.e., the maximum number of codewords, which can be detected from the codeword buffer. Therefore, by decreasing M we may reduce the delay of the decoder. This implies that sometimes the codeword buffer may contain more codewords than we can decode, thus reducing the decoding rate. However, the loss of performance may be negligible since the probability that the codeword buffer contains the maximum number of codewords is low. The number of decoder outputs can be optimized for given application, if statistics of encoded data is available. This approach is used in our MPEG-2 demonstration discussed in the following section. Furthermore, if M is decreased and fixed, we find that the delay of the critical path is constant when $2^{n-1} \leq N < 2^n$ where n is an integer. This implies that the length of the codeword buffer should be chosen such that $N = 2^n - 1$. In this case, MA units are equipped with n full adders.

IV. MPEG-2 Variable Length Decoding Demonstration

The proposed decoding scheme results in a *variable input/variable output rate* system and, therefore, the buffering resources are needed in the input as well as in the output. Our demonstration is targeted to an embedded system assuming external buffering and alignment resources. Hence, only the kernel decoder design consisting of codeword detection and symbol lookup is considered. In order to estimate the performance of the proposed scheme, it has been applied to MPEG-2 video coding standard [18] and this demonstration is described in this section.

A. Requirements

Continuous preprocessed MPEG-2 data strings, which consist only of the variable length code of the discrete cosine transform (DCT) coefficients, have been chosen as the input for our implementation. Several encoded MPEG-2 data streams were analyzed and the obtained statistics are summarized in Table II. This information has been used to derive the requirements for the demonstration system.

The minimum size for the codeword buffer is the length of the longest codeword, i.e., 24 bits in MPEG-2, which implies that the MA units must be equipped with at least five full adders, i.e., $n = 5$. In the demonstration, we have used this minimum requirement. Therefore, the optimum size for the codeword buffer from the critical path point of view is $N = 31$. The 31-bit codeword buffer may contain at most 15 codewords but according to statistics in Table II, 31-bit buffer can contain 5.6 codewords on average and, therefore, the number of decoder outputs, M, can be decreased for shortening the critical path. In our case, the average is rounded upwards and the number of outputs is $M = 6$.

B. Hardware Modeling

The decoder has been described in behavioral-VHDL. Although we target to an FPGA technology, the VHDL description has been kept as technology independent as possible. The structure of demonstrator follows the general organization, i.e., the codeword detection and symbol lookup have been realized

TABLE I
MEMORY ADDRESS GENERATION

Length	Page	Offset
2, 3	000	01001
4	001	1100 & EB(7)
5	010	01 & T & EB(8:9)
6	010	0 & T & EB(6) & EB(8:9)
7	001	01 & T & EB(8:9)
8	010	T & EB(6:9)
9	001	T & EB(6:9)
10	000	01 & EB(8:9) & 0
11	000	0 & T & EB(7:9)
13	000	1 & EB(6:9)
14	011	0 & EB(6:9)
15	011	1 & EB(6:9)
16	100	0 & EB(6:9)
17	100	1 & EB(6:9)

as illustrated in Fig. 3(c) but some MPEG-2-specific modifications were included. These modifications are described in the following.

1) Codeword Detector, CD: The CD unit has at most 12-bit input, which is sufficient to detect all the MPEG-2 codewords, from minimum length of two bits to 24 bits. The CD returns three 6-bit values of a 5-bit codeword length and a 1-bit end-of-block (EOB) status: two values for DC coefficient and one value for ac coefficients. The MPEG-2 standard defines four codeword tables, B.12 – B.15, and the selection of codeword table is controlled by 2-bit VLC control signal *vlcf*, which is the concatenation of the parameters *intra_vlc_format* and *macroblock_intra* defined in [18]. The symbol for the modified CD is depicted in Fig. 4(a).

The input *BitField* is checked for a possible codeword. If detected codeword represents EOB, the EOB status is set "true". In intra decoding, two DC values, *dcl* for luminance and *dcc* for chrominance are returned according to codeword tables B.12 and B.13, respectively. In nonintra decoding, *dcc* represents the value of the DC coefficient. If codeword is not detected from the bit field, zero-lengths are returned and EOB status is maintained as follows. The codeword represents a DC coefficient only if the previous codeword is EOB, thus EOB status is forced to "true." Correspondingly, the codeword is an AC coefficient only if the previous codeword is not EOB and, therefore, EOB status is forced to "false."

2) Chrominance Format Counter, CFC: The CFC is used to select the correct group of the DC candidates out of two possible groups, i.e., chrominance candidates *chrc* and luminance candidates *lumc*. The realization is trivial; a counter returns the chrominance control signal *chr_ctrl* for the next block according to current block number *bnr* in a macroblock as specified in [18]. The maximum block number is controlled by parameter chrominance format *chrf*. The block number is upgraded when EOB is detected. In order to prevent the increase in block number when EOB status is maintained, two previous EOB statuses given with *prEOBs* are checked. The schematic of the CFC is shown in Fig. 4(b) where *DCcs* denotes correct DC candidates.

3) Multiplexed Add, MA: The MA unit is modified to select also between AC *ACcs* and DC candidates *DCcs*. The candidates consist of the values from all the CDs defined by set J_i. The 2–1 multiplexing between AC and DC candidates is controlled by the previous EOB status *EOB* and it can be performed in parallel with the full adder computing the sum of the LSBs of the input operands. The symbol of the modified MA is illustrated in Fig. 4(c). Otherwise, the operation of the MA is similar to the principal operation, i.e., output *nxt_EOB_L* is selected according to the sum *nxt_S* of the previous sum S and the previous length L.

4) Memory Address Generator, MAG: The MAG unit returns an 11-bit *MAG_code*, which may contain memory address or bits that are required for returning the symbol, for each codeword. In order to decode DC coefficient in intra decoding, 11 bits are extracted from the codeword buffer. The bits to be extracted are located according to intermediate sums.

The extracted bits are processed depending on the length and the interpretation of the codeword. If the codeword represents DC coefficient in intra decoding, it specifies the number of bits to be selected according to [18, table B.12 or B.13]. The selected bits are extended to 11-bit *MAG_code* as a two's complement number. Otherwise, the extracted bits contain a complete or partial codeword, which is used to generate the address to the symbol memory.

For describing the memory mapping and address generation method used in the demonstration, let the extracted bits be enumerated from the left to the right and denoted as EB(0:10). Both tables, B.14 and B.15, include at most 16 different codewords of a specific length and consequently, the identification of the codeword requires four bits. However, when combining the codeword tables and mapping them into unified memory, the chosen bits may identify two different codewords depending on the table. In order to distinguish the codewords in different tables, a table bit, T, defined as

$$T = \begin{cases} 1, & \text{if B.15} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

is used to specify the table. Altogether, a 3-bit page as well as the 5-bit offset are generated according to length as shown in Table I. Although, the sign bit is not needed to point the magnitude of the symbol stored into the memory, it should be propagated further for determining the correct level. Therefore, the memory address and sign are embedded into *MAG_code*.

Since only one codeword per cycle can represent symbol ESC in a 31-bit codeword buffer, a shared unit is utilized for extracting ESC and forwarding the 18-bit *ESC_Sym* consisting of possible symbol whose value is not predefined. Similarly, the EOB statuses are propagated further.

5) Symbol Fetch, SF: The symbol fetch consist of three parallel dual-port memory banks and the resources to return the correct symbol. The symbols in tables B.14 and B.15 excluding EOB and ESC are mapped into each memory bank. *MAG_code*s are read in rising clock edge. If the EOB status is true, it is returned and "run" and "level" are forced to zero. If the length of the codeword is equal to 24 implying ESC, a 6-bit "run" followed by 12-bit signed "level" in *ESC_Sym* are returned. For the DC coefficient in intra decoding mode, "run" is forced to zero and *MAG_code* is returned as a "level". Otherwise, the symbol is read from the memory location defined by the address, which is embedded into *MAG_code*. The predefined symbols stored in the memory can be represented with 11 bits, i.e., 5-bit "run" and

TABLE II
PROPERTIES OF MPEG-2 BENCHMARKS AND EXPERIMENTAL RESULTS.

Benchmark	Block type	b	W	B	b/W	W/31b	Scheme C	Scheme W/C	FPGA C	FPGA W/C
bat_327_334	I (B.15)	905 241	172 745	23 298	5.2	5.9	32 104	5.4	33 526	5.2
	NI	1 506 680	266 485	38 940	5.7	5.5	54 843	4.9	56 780	4.7
popplen	I (B.15)	242 795	47 003	4 572	5.2	6.0	8 686	5.4	9 182	5.1
	NI	153 265	28 069	4 139	5.5	5.7	5 555	5.1	5 746	4.9
sarnoff	I (B.14)	429 065	80 563	8 418	5.3	5.8	15 296	5.3	16 198	5.0
	NI	169 567	36 408	8 447	4.7	6.7	6 052	6.0	6 823	5.3
tennis	I (B.14)	57 741	12 345	2 718	4.7	6.6	2 039	6.1	2 339	5.3
	I (B.15)	613 066	120 754	9 504	5.1	6.1	21 721	5.6	22 897	5.3
	NI	989 235	137 756	25 524	7.2	4.3	38 238	3.6	38 682	3.6
t1cheer	I (B.15)	415 873	80 818	8 244	5.1	6.0	14 730	5.5	15 500	5.2
	NI	255 433	51 680	9 432	4.9	6.3	9 113	5.7	9 824	5.3
Total		5 737 961	1 034 626	143 236	5.5	5.6	208 377	5.0	217 497	4.8

b: bits. W: codewords. B: block. b/W: bits per codeword. W/31b: codewords in 31 bits. C: cycles. W/C: codewords per cycle.

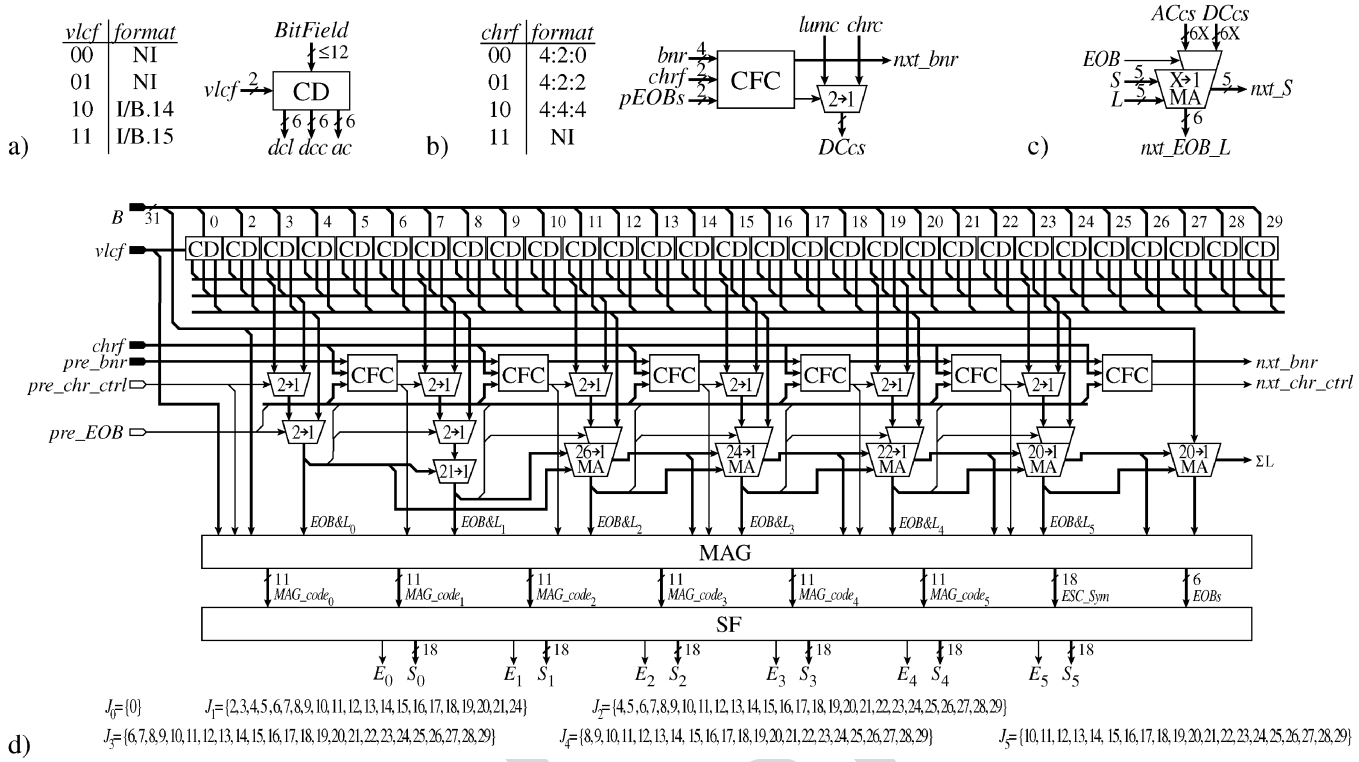

Fig. 4. Block diagrams for MPEG-2 demonstration. (a) Modified CD. (b) Selection of DC coefficient. (c) Modified MA. (d) Entire decoder.

the 6-bit absolute value of "level". Therefore, the "run" is extended to six bits and "level" is converted to 12-bit signed value before returning the actual 18-bit symbol.

6) Entire Decoder: The block diagram of the entire MPEG-2 decoder is illustrated in Fig. 4(d). The codeword detection consists of 29 CD units, which have inputs from buffer locations shown above the CDs. The seven left-most CDs can detect all the possible codewords, next three CDs detect up to 21-bit codewords, and the remaining CDs detect only shorter codewords until the last or the right-most CD detects only 2-bit codewords.

The first valid EOB and length, $EOB\&L_0$, is obtained from the left-most CD but selection between the two DC candidates is needed introducing a 2–1 multiplexer controlled by chrominance control *pre_chr_ctrl* from the previous cycle. Similarly, a 2–1 multiplexer controlled by the EOB status *pre_EOB* from the previous cycle is employed to select between AC and DC candidates. The other $EOB\&L_i$ values are selected from CDs in buffer locations J_i. The correct DC candidates out of luminance and chrominance candidates are selected according the control provided by the corresponding CFC. A 2–1 multiplexer and one 21–1 multiplexer select $EOB\&L_1$ from AC and DC candidates. For the outputs $EOB\&L_2$-$EOB\&L_5$, the modified MA's are used to select valid values. The MA for the third output $EOB\&L_2$ is the most complex having candidates from 26 CDs. Let us remark that the right-most MA is used to provide the extracted bits for the last codeword W_5 and to compute the final sum of the detected codeword lengths.

For the symbol lookup, the variable length coding format *vlcf*, chrominance controls, the EOB statuses, and lengths of the codewords are forwarded to the MAG with the intermediate sums in order to generate the *MAG_codes* for each codeword. Apart from *MAG_codes*, the MAG returns possible escape value *ESC_Sym* and the EOB statuses *EOBs*. During the symbol fetch,

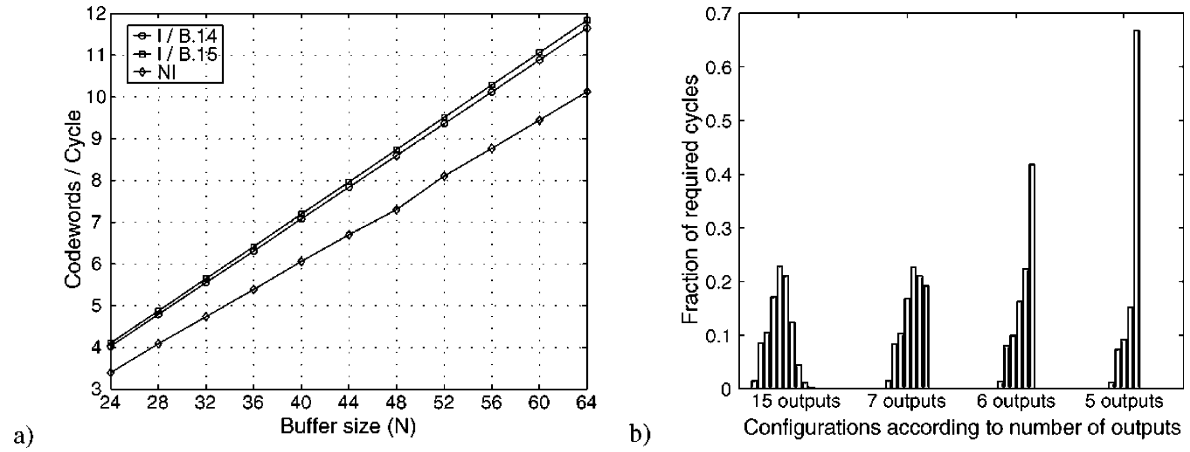

Fig. 5. Experimental results. (a) Throughput of the proposed approach and (b) distribution of symbols over the decoder outputs.

the EOB is interpreted according to the EOB status, which is also returned. The codeword representing the intra DC coefficient is determined from the most significant bit (MSB) of *vlcf* and the EOB status of the preceding codeword. The ESC can be identified from the MSB's of the length. Otherwise, the actual symbol is fetched from the symbol memory.

In general, the MPEG-2-specific modifications are not in the critical path, thus the discussion on decoder delay on the previous section applies to the demonstration. Generation of *MAG_codes*, except the last one, can be performed in parallel with MA's and, therefore, the MAG is not a separate pipeline stage. However, symbol fetch is pipelined since synchronous memories have been used.

V. Experimental Results

The proposed VLD scheme has been experimented with a parametrizable simulation model in Matlab™ and with an FPGA implementation. The simulation model is exploited to analyze the dependencies and behavior of the scheme. The results are given in cycle domain meaning that information on timing or required resources is not considered. The FPGA demonstrator is used to prove the feasibility of the scheme and estimate the hardware complexity. The results are obtained by using Modelsim HDL simulator and Exemplar LeonardoSpectrum. The performance figures of the demonstrator are estimated in time domain.

The highest input rate is obtained when the codeword buffer can be completely updated at each cycle, i.e., if the accumulated length of the complete codewords in the buffer is equal to the buffer size. Assuming such an ideal data stream, the theoretical upper bound for the throughput is equal to buffer size divided by the average codeword length given in column "W/31b" in Table II. In practice, however, the buffer may contain a partial codeword, which cannot be detected at current cycle. Therefore, it should be kept in the buffer and processed at the next cycle when the remaining bits are fetched into the buffer. When applying the proposed scheme to our benchmarks in Table II, the effect of the buffer size to the throughput is illustrated in Fig. 5.

The number of outputs has been decreased in the demonstrator based on statistics and by recognizing the fact that the shorter codewords may not be decoded although they may exist in the buffer. The distribution of the codewords over decoder outputs, i.e., the proportion of cycles returning certain number of symbols, with different decoder configurations is illustrated in Fig. 5(b). The left-most group "15 outputs" represents the theoretical approach, i.e., the scheme with 31-bit codeword buffer and 15 decoder outputs. When experimented with the benchmark data, the proportion of cycles returning more than nine symbols is negligible. Therefore, the experimental results supports the statistical conclusion to decrease the number of the decoder outputs.

In Fig. 5(b), the remarkable drop in proportion can be obtained after seven outputs. The resulting distribution over outputs "7 outputs" is balanced to return from 4 to 7 symbols but, on the other hand, the cycles with the largest proportion are returning five symbols, although with small difference. The balanced proportion between cycles is advantageous if the cycle time is predefined and seven codewords can be detected in the given cycle time. However, the detection of the seventh codeword may increase the critical path and the given cycle time is exceeded.

The distribution with six outputs, noted as "6 outputs", represents our demonstration. The cycles with largest proportion are returning the maximum number of symbols, i.e., six symbols and the difference to the second largest proportion is already remarkable. Furthermore, the most of the cycles are decoding five or six codewords. In order to decode maximum number of codewords during the most of the cycles, the number of outputs should be restricted to five as shown with the group "5 outputs" in Fig. 5(b). When the number of outputs is decreased further, it is obvious that the largest proportion is increasing until symbol-serial decoders are returning one symbol per cycle with proportion of one. However, it should be noted that also the number of cycles required to complete decoding is increased and utilization of codeword buffer is decreased. Altogether, these effects are against our original objective.

The experimental results with scheme and demonstrator in cycle domain are summarized into Table II. Column "Scheme" contains the practical upper bounds for the performance of the scheme with a 31-bit buffer and 15 outputs. The required cycles and achieved throughput for the demonstrator with a 31-bit

buffer and 6 outputs are depicted in column "FPGA." On average, 4.8 codewords per cycle are detected and decoded while the theoretical and practical throughputs in cycle domain are 5.6 and 5.0 codewords per cycle, respectively.

The previous discussion is based on behavioral models and the timing accuracy on unit cycles. However, the critical path defining the cycle time is an important measure for determining the absolute throughput, i.e., the amount of data processed in a time unit. In order to estimate the maximum clock frequency, the VHDL model of the demonstrator has been synthesized on Xilinx Virtex-II™ FPGA (device 2V4000bf957) [19]. The CD units turn out to be application-specific pattern recognizers based on lookup-tables (LUTs). The CFC is also based on LUTs while each MA is synthesized onto a 5-bit ripple carry adder parallel with multiplexer tree. Consequently, the delay of each MA is about the same, i.e., delays of five full adders and one 2–1 multiplexer, although the size of the multiplexer tree varies depending the number of candidates. When the entire design has been synthesized, 2 940 CLBs out of 23 040 were allocated.

Three dual-port Block SelectRAM memories with 160 rows of 11 bits are generated using Xilinx CORE Generator for symbol memories. In an ideal memory mapping, each symbol has location of its own and the number of nonused locations and replicated symbols are zero. In such a case, a 7-bit address space is enough for 111 different predefined symbols. In practise, however, many mapping functions results in nonused locations and some symbols are located in two different locations due to two different codewords representing same symbol. In order to ease the design work, 8-bit address space has been used in the demonstrator. The synthesized design resulted in a critical path of 45.11 ns. The characteristics of the demonstrator are summarized in Table III.

We would like to note that straightforward and fair comparison with other reported decoders is impossible due to different implementation approaches, e.g., different codeword tables, IC technologies (ASIC vs. FPGA), design styles (synchronous vs. asynchronous), and different compression ratios. Finally, we indicate that the proposed scheme is implemented in the prototype MOLEN FPGA processor [20].

TABLE III
CHARACTERISTICS OF MPEG-2 DECODER DEMONSTRATION.

Design platform	Xilinx Virtex II™ FPGA
Application	MPEG-2
CLB count	4 940 / 23 040
Memory	3 dual-port memories with 160 rows of 11 bits
Virtex-II™ specifics	Block SelectRAM™ Memory
Frequency	22 MHz
Throughput	585 Mbits/s 105 Msymbols/s

VI. 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 experimenting and estimating the behavior and performance. It has been shown that the throughput rate of the scheme is proportional to the size of the codeword buffer and, for 31-bit buffer, the average throughput is 5.0 symbols per cycle. The MPEG-2 variable length decoder demonstration has been described in VHDL and mapped onto Xilinx Virtex-II™ FPGA. The evaluated results indicate that 4.8 symbols out of the 5.6 average symbols present in the 31-bit buffer can be detected per cycle. The critical path of 45 ns proves the feasibility and potential of the approach. In the future, we intend to parameterize the demonstrator and concentrate on configurability like different application-specific codeword tables, adaptive coding, and balancing the data flow while decoding several data streams in parallel. Furthermore, data access and buffering techniques need to be studied for using the proposed scheme in MOLEN processor.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July, Oct. 1948.

[2] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, pp. 1098–1101, Sept. 1952.

[3] A. Mukherjee, N. Rangnathan, and M. Bassiouni, "Efficient VLSI designs for data transformation of tree-based codes," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 306–314, Mar. 1991.

[4] R. Hashemian, "Design and hardware implementation of a memory efficient Huffman decoding," *IEEE Trans. Consumer Electron.*, vol. 40, pp. 345–352, Aug. 1994.

[5] 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*, vol. 46, pp. 742–754, June 1999.

[6] S. M. Lei and M. T. Sun, "An entropy coding system for digital HDTV applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 1, pp. 147–155, Mar. 1991.

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

[8] C.-T. Hsieh and S. P. Kim, "A concurrent memory-efficient VLC decoder for MPEG applications," *IEEE Trans. Consumer Electron.*, vol. 42, pp. 439–446, Aug. 1996.

[9] S. Kinouchi and A. Sawada, "Huffman Code Decoding Circuit," U.S. Pat. 5 617 089, Apr. 1, 1997.

[10] 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, Apr. 21–24, 2002.

[11] J. Nikara, S. Vassiliadis, J. Takala, M. Sima, and P. Liuha, "Parallel multiple-symbol variable-length decoding," in *Proc. IEEE Int. Conf. Comput. Design*, Freiburg, Germany, Sept. 16–18, 2002, pp. 126–131.

[12] S. B. Choi and M. H. Lee, "High speed pattern matching for a fast Huffman decoder," *IEEE Trans. Consumer Electron.*, vol. 41, pp. 97–103, Feb. 1995.

[13] M. K. Rudberg and L. Wanhammar, "New approaches to high speed Huffman decoding," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Atlanta, GA, May 1996, pp. 149–152.

[14] 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.*, vol. 11, pp. 210–221, Feb. 2001.

[15] 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*, Austin, TX, Sept. 24–26, 2001, pp. 425–430.

[16] B. W. Y. Wei and T. H. Meng, "A parallel decoder of programmable Huffman codes," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 175–178, Apr. 1995.

[17] C.-J. Chang, S. Vassiliadis, and J. G. Delgado-Frias, "An investigation of binary CLA and ripple CMOS adder designs," *Microprocess. Microprogr. J.*, vol. 40, no. 1, pp. 1–21, Jan. 1994.

[18] *Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Video, ITU-T Recommendation H.262*, International Telecommunication Union, 2000.

[19] *Virtex-II Platform FPGA Handbook*, UG002 (v1.0) ed., Xilinx, Inc., 2000.

[20] S. Vassiliadis, S. Wong, and S. Cotofana, "The MOLEN $\rho\mu$-coded processor," in *Proc. Int. Conf. Field-Programmable Logic Applications*, Belfast, Northern Ireland, U.K., Aug. 27–29, 2001, pp. 275–285.

Jari Nikara (S'XX) **<<Author: Please provide membership year information, thank you>>** received the M.Sc. degree (with distinction) in information technology from Tampere University of Technology, Tampere, Finland, in 2000. He is currently working toward the Dr.Tech. degree at the same university.

In 1999, he was a Research Assistant with the Signal Processing Laboratory, Tampere University of Technology. Since 2000, he has been a Researcher at the Signal Processing Laboratory and Institute Digital and Computer Systems, Tampere University of Technology. In 2001, he was a Visiting Researcher at the Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands. His research interests include multimedia hardware accelerators.

Stamatis Vassiliadis (F'XX) **<<Author: Please provide membership year information, thank you>>** is currently a Chair Professor in the Electrical Engineering Department of Delft University of Technology (TU Delft), The Netherlands. He has also been a Faculty Member on the Electrical Engineering Departments of Cornell University, Ithaca, NY and the State University of New York (S.U.N.Y.), Binghamton, NY. He was with IBM for ten years **<<Author: Please provide city/state information for IBM, thank you>>** where he was involved in a number of advanced research and development projects. For his work, he received numerous awards including 24 publication awards, 15 invention awards and an outstanding innovation award for engineering/scientific hardware design. He has 70 U.S. patents, which rank him as the top all time IBM inventor.

In 1992 he received an Honorable Mention Best Paper Award from the ACM/IEEE MICRO25. He received the Best Paper Awards in the IEEE CAS, in 1998 and 2002, IEEE ICCD in 2001, and PDCS in 2002.

Jarmo Takala (SM'XX) **<<Author: Please provide membership year information, thank you>>** received the M.Sc. degree (with distinction) in electronics and the Dr.Tech. degree in information technology from Tampere University of Technology (TUT) , Tampere, Finland, in 1987 and 1999, respectively.

From 1992 to 1996, he was a Research Scientist with VTT-Automation, Tampere, Finland. Between 1995 and 1996, he was a Senior Research Engineer at Nokia Research Center, Tampere, Finland. From 1996 to 1999, he was a Researcher at TUT. Currently, he is a Professor in the Department of Computer Engineering at TUT. His research interests include circuit techniques, parallel structures, and design methodologies for digital signal processing systems.

Petri Liuha received the M.Sc. degree in information technology from Tampere University of Technology, Tampere, Finland, in 1992.

From 1991 to 1992, he was an Research and Development Engineer with Nokia Consumer Electronics **<<Author: Please provide city/state information for Nokia, thank you>>**. In 1993, he joined Nokia Research Center, Tampere, Finland, where he has worked as a Research Engineer in different areas of implementation of video signal processing and multimedia. Currently, he is a Research Manager of the Media Processors Group. His current research interests are in architectural developments for implementations of multimedia applications.