A Reconfigurable Functional Unit for TriMedia/CPU64. A Case Study

Mihai Sima[1,2], Sorin Cotofana[1], Stamatis Vassiliadis[1],
Jos T.J. van Eijndhoven[2], and Kees Vissers[3]

[1] Delft University of Technology, Department of Electrical Engineering,
Mekelweg 4, 2628 CD Delft, The Netherlands,
{M.Sima,S.D.Cotofana,S.Vassiliadis}@et.tudelft.nl
[2] Philips Research Laboratories, Department of Information and Software Technology,
Professor Holstlaan 4, 5656 AA Eindhoven, The Netherlands,
jos.van.eijndhoven@philips.com
[3] TriMedia Technologies, Inc., 1840 McCarthy Boulevard, Milpitas, California 95035, U.S.A.,
kees.vissers@trimedia.com

Abstract. The paper presents a case study on augmenting a TriMedia/CPU64 processor with a Reconfigurable (FPGA-based) Functional Unit (RFU). We first propose an extension of the TriMedia/CPU64 architecture, which consists of a RFU and its associated instructions. Then, we address the computation of the 8×8 IDCT on such extended TriMedia, and propose a scheme to implement an 8-point IDCT operation on the RFU. Further, we address the decoding of Variable Length Codes and describe the FPGA implementation of a Variable Length Decoder (VLD) computing facility. When mapped on an ACEX EP1K100 FPGA from Altera, our 8-point IDCT exhibits a latency of 16 and a recovery of 2 TriMedia cycles, and occupies 42% of the FPGA's logic array blocks. The proposed VLD exhibits a latency of 7 TriMedia cycles when mapped on the same FPGA, and utilizes 6 of its embedded array blocks. By using the 8-point IDCT computing facility, an 8×8 IDCT including all overheads can be computed with the throughput of 1/32 IDCT/cycle. Also, with the proposed VLD computing facility, a single DCT coefficient can be decoded in 11 cycles including all overheads. Simulation results indicate that by configuring each of the 8-point IDCT and VLD computing facilities on a different FPGA context, and by activating the contexts as needed, the augmented TriMedia can perform MPEG macroblock parsing followed up by a pel reconstruction with an improvement of 20-25% over the standard TriMedia.

1 Introduction

A common issue addressed by computer architects is the range of performance improvements that may be achieved by augmenting a general purpose processor with a reconfigurable core. The basic idea of such approach is to exploit both the general purpose processor capability to achieve medium performance for a large class of applications, and FPGA flexibility to implement application-specific computations. Thus far FPGA-augmented processors have predominantly assumed a simple general purpose core [1–4]. Considering the class of VLIW machines, two general research questions may be raised:

– What are the influences of reconfigurable arrays on the performance of *commercially available* VLIW processors?
– What are the architectural changes needed for incorporating the reconfigurable array into the processor core?

In an attempt to answer to these questions, we will present a case study on augmenting a TriMedia/CPU64 processor with a Reconfigurable (FPGA-based) Functional Unit (RFU). With such RFU, the user is given the freedom to define and use any computing facility subject to the FPGA size and TriMedia/CPU64 organization. In order to evaluate the potential performance of the augmented TriMedia/CPU64, we chose a significant chunk of MPEG decoding as benchmark. In particular, since the video data accounts for more than 80% of the whole MPEG bit stream [5], we considered the parsing of Variable-Length (VL) coded data at the macroblock layer followed by a pel reconstruction procedure as benchmark. That is, all the *data elements* corresponding to slice and higher layers are considered as being constants for our experiment.

We decided to provide hardware support for two functions of the selected benchmark: 8-point (1-D) Inverse Discrete Cosine Transform (IDCT) and Variable-Length Decoder (VLD). By developing VHDL code and mapping it with Altera tools, we evaluated the performance of these FPGA-based functions. Further, a program which is MPEG-compliant has been written in C, and then compiled, scheduled and finally simulated with TriMedia tool-chain. For a typical MPEG string with 10% intra-coded, 70% B-coded, and 20% P-coded macroblocks, we found that the augmented TriMedia/CPU64 can perform macroblock parsing followed up by a pel reconstruction with an improvement of 20-25 % over the standard TriMedia. Given the fact that TriMedia/CPU64 is a 5 issue-slot VLIW processor with 64-bit datapaths and a very rich multimedia instruction set, such an improvement within the target media processing domain indicates that the hybrid TriMedia/CPU64 + FPGA is a feasible approach.

The paper is organized as follows. For background purposes, we briefly present several issues concerning MPEG and the FPGA architecture in Section 2. Section 3 describes the architectural extension of TriMedia/CPU64. Implementation issues related to 1-D IDCT and VLD computing facilities and their corresponding instructions are discussed in Sections 4 and 5. The 8×8 IDCT and entropy decoder implementations are then described in Sections 6 and 7. The execution scenario of the chosen benchmark on both standard and extended TriMedia, and experimental results are presented in Section 8. Section 9 completes the paper with some conclusions and closing remarks.

2 Background

Data compression is the reduction of redundancy in data representation, carried out for decreasing data storage requirements and data communication costs. A typical video codec system is presented in Figure 1 [6, 5]. The lossy source coder performs filtering, transformation (such as DCT, subband decomposition, or differential pulse-code modulation), quantization, etc. The output of the source coder still exhibits various kinds of statistical dependencies. The (lossless) entropy coder exploits the statistical properties of data and removes the remaining redundancy after the lossy coding.

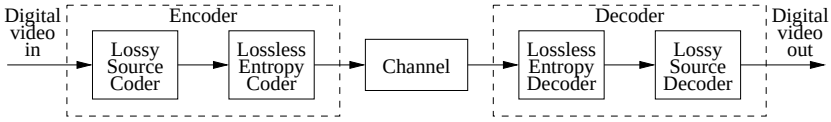


Fig. 1. The block diagram of a generic video codec – adapted from [6, 5].

In MPEG, the DCT-Quantization pair is used as a lossy coding technique. The DCT algorithm processes the video data in blocks of 8×8, decomposing each block into a weighted sum of 64 spatial frequencies. At the output of DCT, the data is also organized in 8×8 blocks of coefficients, each coefficient representing the contribution of a spatial frequency for the video block being analyzed. Since the human eye cannot readily perceive high spatial frequency activity, a quantization step is carried out. The goal is to force as many DCT coefficients as possible to zero, especially those corresponding to high spatial frequencies, within the boundaries of the prescribed video quality. Then, a zig-zag operation transforms the matrix into a vector in which the coefficients are ordered from the lowest frequencies (upper-left hand corner of the 8×8 block) to the higher ones (lower-right hand corner of the matrix). Usually, this vector exhibits large numbers of consecutive zeros. The subsequent compression step is carried out by the entropy coder which consists of two major parts: Run-Length Coder (RLC) and Variable-Length Coder (VLC). The RLC represents consecutive zeros by their run lengths. Since not each and every zero is coded, the number of samples is reduced. The RLC output data are composite words, also referred to as *source symbols*, which describe pairs of zero-run lengths and quantized DCT coefficient values. When all the remaining coefficients in a vector are zero, they are all coded by the special symbol *end-of-block*. Variable length coding, also known as Huffman coding, is a mapping process between source symbols and *variable length codewords*. The variable length coder assigns shorter codewords to frequently occuring source symbols, and vice versa, so that the average bit rate is reduced. In order to achieve maximum compression, the coded data is sent through a continuous stream of bits with no specific guard bit assigned to separate between two consecutive symbols. As a result, decoding procedure must recognize the code length as well as the symbol itself in this case.

Subsequently, we will focus on the MPEG decoding, i.e., on the inverse operation of MPEG coding. Further, we will briefly present the theoretical background of Inverse Discrete Cosine Transform (IDCT), entropy decoding, as well as some issues related to the MPEG standard.

2.1 Inverse Discrete Cosine Transform

The transformation for an N point 1-D IDCT is defined by [7]:

$$x_i = \frac{2}{N} \sum_{u=0}^{N-1} K_u X_u \cos \frac{(2i+1)u\pi}{2N}$$

where X_u are the inputs, x_i are the outputs, and $K_u = \sqrt{1/2}$ for $u = 0$, otherwise is 1. For MPEG, a 2-D IDCT processes an 8×8 matrix X [5]:

$$x_{i,j} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} K_u K_v X_{u,v} \cos\frac{(2i+1)u\pi}{16} \cos\frac{(2j+1)v\pi}{16}$$

One strategy to compute the 2-D IDCT is the standard row-column separation. The 2-D transform is performed by applying the 1-D transform to each row (horizontal IDCTs) and subsequently to each column (vertical IDCTs) of the data matrix. This strategy can be combined with different 1-D IDCT algorithms to further reduce the computational complexity. One of the most efficient 1-D IDCT algorithms has been proposed by Loeffler [8]. A slightly different version of the Loeffler algorithm in which the $\sqrt{2}$ factors are moved around has been proposed by van Eijndhoven and Sijstermans [9]. In our experiment, we will use this modified algorithm (see Figure 2).

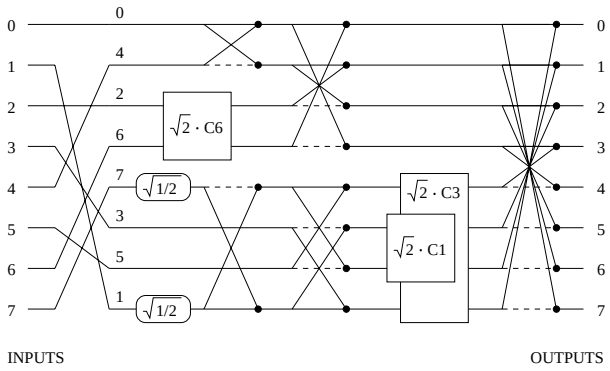
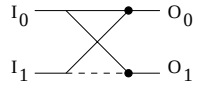


Fig. 2. The modified 'Loeffler' algorithm – from [9].

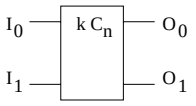
In the Figure, the round block signifies a multiplication by $C'_0 = \sqrt{1/2}$. The butterfly block and the associated equations are presented in Figure 3.



$$O_0 = I_0 + I_1$$
$$O_1 = I_0 - I_1$$

Fig. 3. The butterfly – from [8].

A square block depicts a rotation which transforms a pair $[I_0, I_1]$ into $[O_0, O_1]$. The symbol of a rotator and the associated equations are presented in Figure 4. Although an implementation of such a rotator with three multiplications and three additions is possible [8], we use the direct implementation of the rotator with four multiplications and two additions, since it shortens critical path and improves numerical accuracy. Therefore, multiplications by constants C'_0, C'_1, S'_1, C'_3, S'_3, C'_6, and S'_6 have to be carried out. For more details regarding this problem, we refer the reader to the bibliography [10].



$$O_0 = I_0 k \cos\tfrac{n\pi}{16} - I_1 k \sin\tfrac{n\pi}{16} = C'_n I_0 - S'_n I_1$$

$$O_1 = I_0 k \sin\tfrac{n\pi}{16} + I_1 k \cos\tfrac{n\pi}{16} = S'_n I_0 + C'_n I_1$$

Fig. 4. The rotator – [8].

2.2 Entropy Decoder

In MPEG, the entropy decoder consists a Variable-Length Decoder (VLD) followed by a Run-Length Decoder (RLD). The input to the VLD is the incoming encoded bit stream, and the output is the decoded symbols. Since the code length of the symbol is variable, both the input and output bit rate of a VLD cannot be kept constant. Three different decoder types are possible [6]: constant input rate, constant output rate, and variable input-output rate.

The *constant-input-rate* VLD decodes a fixed number of bits and produces a variable number of symbols per unit time. An example of such decoder which decodes one bit per cycle is described in [11]. The decoder employs a binary tree search technique in which a token is propagated in a reverse Huffman tree constructed from the original codes. Although some improvements of the tree-based method make it possible to decode more than one bit per cycle [12], the tree-based approaches are not suitable for high performance applications such as high-definition television, because high clock rate processing is needed.

A *constant-output-rate* VLD decodes one codeword (symbol) per cycle regardless of its length [13]. Generally speaking, a constant-output-rate VLD contains a look-up table which receives the variable-length code itself as the address. The decoded symbol (run-level pair or end-of-block) and the codeword length are generated in response to that address. Since the longest codeword excluding Escape has 17 bits, the LUT size could reach 131072 ($= 2^{17}$) words for a direct mapping of all possible codewords.

A *variable-input-output-rate* VLD is a mixture of the first two VLDs. It is implemented as a repeated table look-up, each step decoding a variable size chunk of bits. If a valid code was encountered, a run/level pair or an end-of-block is generated. If a miss is detected, a chunk size for the next look-up is generated. In this way, the short (most probable) are preferentially decoded. A variable-input-output-rate VLD exhibits an acceptable decoding throughput, while the size of the look-up table is resonable small.

The run-length decoder passes the VLC-decoded codewords through if they are not run-length codes, otherwise it outputs the specified number of zeros.

2.3 Macroblock parsing and pel reconstruction

The macroblock parsing process reads the VL coded data string from which all the headers corresponding to slice and higher layers have been removed, and outputs various symbols: *decoding parameters* at the macroblock layer (*macroblock_address_increment*, *macroblock_type*, *coded_block_pattern*, and *quantizer_scale*), *motion values*, and *composite symbols* (*run/level* pairs and *end_of_block*). The decoding of the Variable-Length Codes (VLC) is performed according to a set of VLC tables defined by the MPEG

standard. The motion values are used by a motion compensation process which is not considered here. However, since these values are decoded during the macroblock parsing, the overhead associated with the decoding of the motion values will be taken into consideration in the subsequent experiment.

Following the macroblock parsing, a pel reconstruction process recreates 8×8 matrices of pels. The pel reconstruction module is depicted in Figure 5. Its functionality is as follows. First, 8×8 matrices of DCT quantized coefficients are recreated by a Matrix Reconstruction module. Second, an inverse quantization (InvQ) is performed. An 8×8 quantization table, and a multiplicative quantization factor (*quantizer_scale*) are used in the InvQ process. Third, a DC prediction unit reconstructs the DC coefficient in intra-coded macroblocks. Finally, an IDCT is performed. In connection with Figure 5 and the subsequent experiment, we would like to mention that the VLC decoder and IDCT will benefit from reconfigurable hardware support.

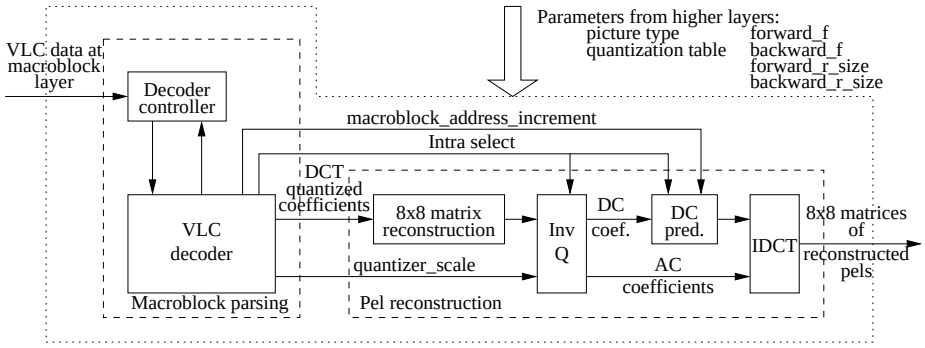


Fig. 5. Macroblock parsing and pel reconstruction module – adapted from [5].

We conclude this section with a review on the architecture of the FPGA we used as an experimental reconfigurable core.

2.4 The FPGA architecture.

Field-Programmable Gate Arrays (FPGA) [14] are devices which can be configured *in the field* by the end user. In a general view, an FPGA is composed of two constituents: *Raw Hardware* and *Configuration Memory*. The function performed by the raw hardware is defined by the information stored into the configuration memory. Generally speaking, a multiple-context FPGA [15] is an FPGA having the configuration memory replicated in order to contain several configurations for the raw hardware. That is, a multiple-context FPGA contains an on-chip cache of raw hardware configurations, which are referred to as *contexts*. Such a cache allows a context switch to occur on the order of nanoseconds [16]. However, loading a new configuration from off-chip is still limited by low off-chip bandwidth.

In the sequel, we will assume that the architecture of the raw hardware is identical with that of an ACEX 1K device from Altera [17]. Our choice could allow future single-

chip integration, since both ACEX 1K FPGAs and TriMedia are manufactured in the same TSMC technological process. Briefly, an ACEX 1K device contains an array of Logic Cells, each including a 4-input Look-Up Table (LUT), a relative small number of Embedded Array Blocks, each EAB being actually a RAM block with 8 inputs and 16 outputs, and an interconnection network. In order to have a general view, we mention that the logic capacity of the ACEX 1K family ranges from 576 logic cells and 3 EABs for EP1K10 device to 4992 logic cells and 12 EABs for EP1K100 device. The maximum operating frequency for synchronous designs mapped on an ACEX 1K FPGA is 180 MHz. More details regarding the architecture and operating modes of ACEX 1K devices, as well as data sheet parameters can be found in [17].

3 An architectural extension for TriMedia/CPU64

TriMedia/CPU64 is a 64-bit 5 issue-slot VLIW core [18], launching a long instruction every clock cycle. It has a uniform 64-bit wordsize through all functional units, the register file, load/store units, on-chip highway and external memory. Each of the five operations in a single instruction can (in principle) read two register arguments and write one register result. The architecture supports subword parallelism and is optimized with respect to media processing. With the exception of floating point divide and square root, all functional units have a recovery[1] of 1, while their latency[2] ranges from 1 to 4. The TriMedia/CPU64 VLIW core also supports multi-slot operations, or super-operations. Such a super-operation occupies two neighboring slots in the VLIW instruction, and maps to a double-width functional unit. This way, operations with more than 2 arguments and/or more than one result are possible.

First we propose that the TriMedia/CPU64 processor is augmented with a Reconfigurable Functional Unit (RFU) which consists mainly of a multiple-context FPGA core. A hardwired Configuration Unit which manages the reconfiguration of the raw hardware is associated to the reconfigurable functional unit, as it is depicted in Figure 6. The reconfigurable functional unit is embedded into TriMedia as any other hardwired functional unit is, i.e., it receives instructions from the instruction decoder, reads its input arguments from and writes the computed values back to the register file. In this way, only minimal modifications of the basic architecture are required.

In order to use the RFU, a kernel of new instructions is needed. This kernel constitutes the extension of the TriMedia/CPU64 instruction set architecture we propose. It includes the following instructions: `SET_CONTEXT`, `ACTIVATE_CONTEXT`, and `EXECUTE`. Loading a context information into the RFU configuration memory is performed under the command of a `SET_CONTEXT` instruction. The `ACTIVATE_CONTEXT` instruction controls the swaping of the active configuration with one of the idle on-chip configuration. The operations performed by the computing resources configured on the raw hardware are launched by `EXECUTE` instructions. In this way, the execution of an RFU-mapped operation requires three basic stages: set, activate, and execute [19].

The user is given a number of `EXECUTE` instructions which encompass different operation patterns: single- or double-slot operations, operations with an immediate ar-

[1] Minimum number of clock cycles between the issue of successive operations.
[2] Clock cycles between the issue of an operation and availability of its results.

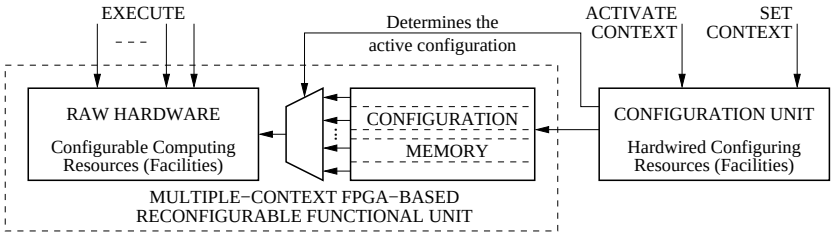


Fig. 6. The organization of the RFU and associated configuration unit.

gument, etc. It is the responsibility of the user to choose the appropriate `EXECUTE` instruction corresponding to the pattern of the operation to be executed. At the source code level, this may be done setting up an *alias*, as it is described subsequently. Since the `EXECUTE` instructions are executed on the RFU without checking of the active configuration, it is still the responsibility of the user to perform the management of the active and idle configurations.

For the semantics of an operation performed by a computing facility, its latency, recovery, and slot assignment are all user definable, the source code of the application should contain information to augment the Machine Description File [20]. Assuming for example a user-defined `VLD` instruction, a way to specify such information is to annotate the source code as follows:

```
.alias      VLD     EXEC_3  ; specifies the alias EXECUTE_3
                            ; (super-op with two inputs and outputs)
.latency    VLD     7       ; specifies the VLD latency
.recovery   VLD     7       ; specifies the VLD recovery
.slot       VLD     1+2     ; specifies the slot assignment
                            ; of the VLD instruction
```

In a similar way, the user can define as many RFU-related instructions as he/she wants.

The next section will present the sintax and semantics of the 1-D IDCT and VLD instructions, as well as implementation issues of the corresponding computing facilities.

4 1-D IDCT instruction and computing facility

Since the standard TriMedia provides a good support for transposition and matrix storage, we expect to get little benefit if we configure the entire 2-D IDCT into FPGA. Our goal is to balance the cost of storing the intermediate 2-D IDCT results into an FPGA-resident transpose matrix memory against obtaining free slots into TriMedia. Consequently, only a super-operation computing the 1-D IDCT of eight 16-bit values packed in two 64-bit registers is considered. The sintax of such operation is:

1-D_IDCT Rx, Ry → Rz, Rw

where the registers Rx and Ry specify the inputs, and Rz and Rw, the outputs. All registers Rx, Ry, Rz, and Rw encompass the common format presented in Table 1.

Table 1. 1-D_IDCT – The common format of registers Rx, Ry, Rz, and Rw (`vec64sh`).

Field name	Acronym	Width (bit)	Position (bit)	Type (TriMedia)	Range	Description
1^{st} value	–	16	$63 \ldots 48$	int16	–	–
2^{nd} value	–	16	$47 \ldots 32$	int16	–	–
3^{rd} value	–	16	$31 \ldots 16$	int16	–	–
4^{th} value	–	16	$15 \ldots 0$	int16	–	–

Since there are no dependencies in computing the 1-D IDCT on each row (column) of the 8×8 matrix, a pipelined 1-D IDCT is desirable. A recovery of 1 of such computing resource implies that the FPGA clock frequency is equal with the TriMedia clock frequency. Nowadays, the current TriMedia clock frequency is greater than 200 MHz, while the maximum allowable clock frequency for ACEX 1K is 180 MHz. Therefore, an 1-D IDCT hypothetical implementation having a recovery of 1 is not a realistic scenario, and a recovery of 2 or more is mandatory for the time being. In the sequel, we will assume a recovery of 2 for 1-D IDCT and a 200 MHz TriMedia. This implies that the pipelined implementation of 1-D IDCT will work with 100 MHz clock frequency.

All the operations required to compute 1-D IDCT are implemented using 16-bit fixed-point arithmetic. Since an implementation of the rotator with four multiplications is preferred [10], the computation of 1-D IDCT requires 14 multiplications. As all the multiplications are to be performed in parallel, an efficient implementation of each multiplication is of crucial importance. For all multiplications, the multiplicand is a 16-bit signed integer represented in 2's complement notation, while the multiplier is a positive integer constant of 15 bits or less. As claimed in [21], these word lengths in connection with fixed-point arithmetic are sufficient to fulfill the IEEE numerical accuracy for IDCT in MPEG applications [22].

A general multiplication scheme for which both multiplicand and multiplier operands are unknown at the implementation time exhibits the largest flexibility at the expenses of higher latency and larger area. If one of the operands is known at the implementation time, the flexibility of the general scheme becomes useless, and a customized implementation of the scheme will lead to improved latency and area. A scheme which is optimized against one of the operands is referred to as *multiplication-by-constant*. Since such a scheme is more appropriate for our application, we will use it subsequently.

To implement the multiplication-by-constant scheme, we built a partial product matrix, where only the rows corresponding to a '1' in the multiplier operand are filled in. Then, reduction schemes which fit into a pipeline stage running at 100 MHz are sought. It should be emphasized that a reduction algorithm which is optimum on a certain FPGA family may not be optimum for a different family.

In connection with the partial product matrix, reduction modules which can run at 100 MHz when mapped on an ACEX 1K are presented in Figure 7. All the designs are synchronous, i.e., both inputs and outputs are registered. The estimations have been obtained by compiling VHDL source codes with Leonardo Spectrum^{TM} from Exemplar, followed by a place and route procedure performed by MAX+PLUS II^{TM} from Altera. The 100 MHz reduction modules are summarized below:

- Horizontal reductions of three, or four 16-bit lines to one line (Fig. 7 – a).
- Horizontal reduction of only two 30-bit lines to one line (Fig. 7 – b).
- Vertical reductions of three or four 7-bit columns to one line (Fig. 7 – c).
- Vertical reductions of six 5- or 6-bit columns to one line (Fig. 7 – d).

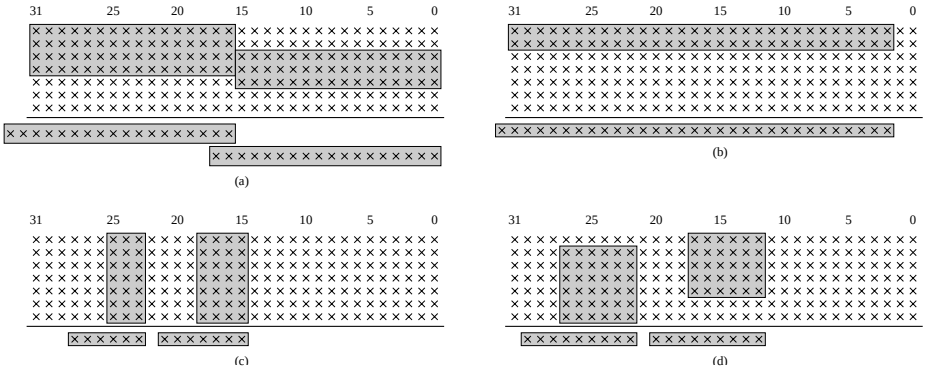


Fig. 7. 100 MHz reduction modules on ACEX 1K.

We do not go into details about the implementations of the multipliers and we refer the reader to [10]. We still mention the latency of each multiplier: $\times C'_0$ latency $= 2$, $\times C'_1$latency $= 3$, $\times S'_1$ latency $= 3$, $\times C'_3$ latency $= 3$, $\times S'_3$ latency $= 3$, $\times C'_6$ latency $= 3$, $\times S'_6$ latency $= 2$.

The sketch of the 1-D IDCT pipeline is depicted in Figure 8 (the Roman numerals specify the pipeline stages). Considering the critical path, the latency of the 1-D IDCT is composed of:

- one TriMedia cycle for reading the input operands from the register file into the input flip-flops of the 1-D IDCT computing resource;
- two FPGA cycles for computing the multiplication by constant C'_0;
- one FPGA cycle for computing all additions to rotators $\sqrt{2}C_1$ and $\sqrt{2}C_3$.
- three FPGA cycles for computing the multiplication by constant C'_1;
- one FPGA cycle for computing the additions in the last stage of the transform;
- one TriMedia cycle for writing back the results from the output flip-flops of the 1-D IDCT computing resource into the register file.

Therefore, the latency of the 8-point 1-D IDCT operation is $1+(2+1+3+1)\times 2+1 = 16$ TriMedia cycles. We evaluated that 1-D IDCT uses 42% of the logic elements of an ACEX EP1K100 device and 257 I/O pins.

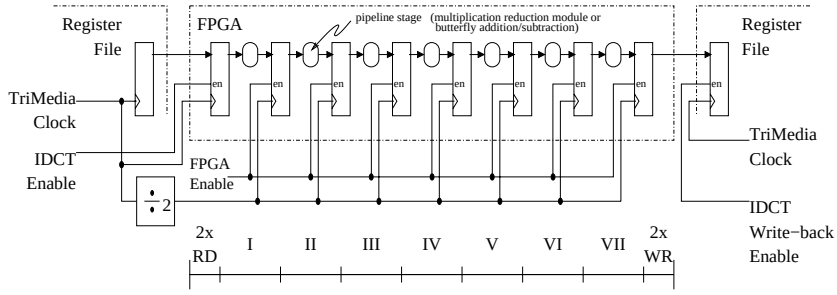


Fig. 8. The 1-D IDCT pipeline

5 VLD instruction and computing facility

As mentioned in Section 3, computing resources which can perform rather complex operations are worth to be implemented on the RFU. Also, as with all hardwired computing resources, the latency of an RFU-configured computing resource should be known at compile time. Therefore, we will subsequently consider a VLD instruction which returns a DCT symbol (*run/level* pair or *end-of-block*) per execution. That is, a constant-output-rate VLD is to be employed. With such decoder, no benefits from preferentially decoding the short (most probable) codewords can be achieved.

A super-operation pattern with two input (Rx, Ry) and two output (Rz, Rw) registers is assigned to the variable-length decoder:

$$\textbf{VLD} \quad \textbf{Rx, Ry} \rightarrow \textbf{Rz, Rw}$$

Table 2. VLD-1 – The format of the first argument (parameter) register – Rx (`uint32`).

Field name	Acronym	Width (bit)	Position (bit)	Type (TriMedia)	Range	Description
Decoding parameters	dec_param	32	31 . . . 0	uint32	–	
Not used	–	27	31 . . . 5	n.a.	n.a.	
MPEG standard	mpeg_s	1	4	bit	$\{0, 1\}$	= 1 for MPEG-2
Intra VLC format	i_vlc_f	1	3	bit	$\{0, 1\}$	= 0 for B14 table
Intra/PB	intra_pb	1	2	bit	$\{0, 1\}$	= 1 for intra macroblock
Luma/Chroma	y_c	1	1	bit	$\{0, 1\}$	= 1 for luminance
DC/AC Coefficient	dc_ac	1	0	bit	$\{0, 1\}$	= 1 for DC coefficient

The Rx register specifies the decoding parameters which identify the type of the symbol to be decoded: AC/DC, luminance/chrominance, intra/non-intra, as well as whether the string is an MPEG-1 or MPEG-2 one, or whether the decoding table is B14 or B15 [5]. The second register, Ry, contains 64 bits of the VL compressed data. The decoded symbol and its code length will be stored into registers Rz and Rw, respectively. Since

Table 3. VLD-1 – The format of the second argument register – Ry (`uint64`).

Field name	Acronym	Width (bit)	Position (bit)	Type (TriMedia)	Range	Description
MPEG string	–	64	$63 \ldots 0$	`uint64`	n.a.	The first bit of the MPEG string is the most-significant bit

Table 4. VLD-1 – The format of the returned value in register Rz (`vec64ub`).

Field name	Acronym	Width (bit)	Position (bit)	Type (TriMedia)	Range	Description
Not used		32	$63 \ldots 32$	*any*	n.a.	
Level	level	16	$31 \ldots 16$	int16	–	Extracted as two uint8.
Run	run	8	$15 \ldots 8$	uint8	–	
Code-length	code_length	8	$7 \ldots 0$	uint8	–	

Table 5. VLD-1 – The format of the returned value in register Rw (`vec64ub`).

Field name		Acronym	Width (bit)	Position (bit)	Type (TriMedia)	Range	Description
Not used		–	32	$63 \ldots 32$	*any*	n.a.	
Not used		–	8	$31 \ldots 24$	uint8	n.a.	
`Exit` controls		–	8	$23 \ldots 16$	uint8	–	
	valid_decode	valid_decode	1	19	bit	$\{0, 1\}$	$= 1$ when valid decode
	error	error	1	18	bit	$\{0, 1\}$	$= 1$ when error
	EOB	EOB	1	17	bit	$\{0, 1\}$	$= 1$ when end-of-block
	exit_flag	exit_flag_1	1	16	bit	$\{0, 1\}$	$= 1$ when exit condition
Not used		–	8	$15 \ldots 8$	uint8	n.a.	
`Exit` flag		–	8	$7 \ldots 0$	uint8	$\{0, 1\}$	
	exit_flag	exit	1	1	bit	$\{0, 1\}$	$= 1$ exit condition

the VLD does not know the start of the next variable-length codeword until the current codeword is decoded, a new **VLD** operation can be launched only after the previous one has completed. Consequently, a recovery lower than the latency gives no advantages, and such implementation should not be sought. The formats of the registers Rx, Ry, Rz, Rw are shown in Tables 2, 3, 4, and 5.

Generally speaking, a constant-output-rate VLD computes the codeword length by looking-up the 17 leading bits of the incoming bit stream into a look-up table. The decoder then sends the code length and the leading bits to other feed-forward circuitry for further decoding and immediately shifts the input by a number of bits equal with *code length*, to prepare the next decoding cycle. In cases where the number of codewords is large, there are some bits that are common to the long VLC's, called *prefix*. By exploiting these common prefixes, the size of the LUT can be reduced because the prefixes are no longer redundant in the LUT [23, 24]. The basic idea of prefix precoding is to group

the VLC's by their common prefixes, and to provide for LUTs, one for each group, which can decode codewords only in the corresponding group.

Since a single EAB of an ACEX 1K device can implement a lookup table of 8 inputs, we partitioned the VLC table according to this FPGA architectural characteristic, as presented in Table 6.

Table 6. The partitioning of the VLC codes of AC coefficients into groups and classes.

Name of the group	No. of symbols in the class	Class / Leading bit-sequence	Code length	Bypassed bit-sequence	Effective address length
DC Group 0	2	1	1 + s	–	n.a.
End-of-block	1	10	2	–	n.a.
AC Group 0	2	11	2 + s	–	n.a.
Escape	1	0000 01	6 + 18/(14,22)	–	n.a.
Group 1	2	011	3 + s		3
	4	010	4 + s		4
	4	0011	5 + s		5
	2	0010 1	5 + s	0	5
	8	0001	6 + s		6
	8	0000 1	7 + s		7
	16	0010 0	8 + s		8
Group 2	16	0000 001	10 + s		5
	32	0000 0001	12 + s	0000 00	7
	32	0000 0000 1	13 + s		8
Group 3	32	0000 0000 01	14 + s		6
	32	0000 0000 001	15 + s	0000 0000 0	7
	32	0000 0000 0001	16 + s		8

In order to reduce the latency, the implementation of the VLD makes use of advanced computation. The run and level for each and every group were decoded in parallel, as the valid symbol would belong to that group. In parallel, the code length of the symbol along with some *selection signals* are determined. Then, the selection of the proper run and level pair is carried out. The implementation is presented in Figure 9.

Regarding the groups 1, 2, and 3, one, six, and nine leading bits are shifted out from the original VLC string, respectively. The three resulted strings are each sent to a different EAB, and three run/level pairs are generated as if the shifted leading bits would have been those mentioned in the column *Bypassed header*. By means of combinatorial circuits, the same procedure is carried out for groups 0, end-of-block, and escape.

Each of the leading bit-sequence which define the VLC class is decoded by a multiple-input gate. Once the class is detected, a multiplexer will select the proper output from the outputs of EABs, EOB detector, Escape detector, and Group 0 decoding. The code length of the decoded symbol is generated according to the detected class.

By simulation, we found that the FPGA-based VLD operation exhibits a latency of 7 TriMedia cycles. 6 EABs of an ACEX EP1K100 device are used.

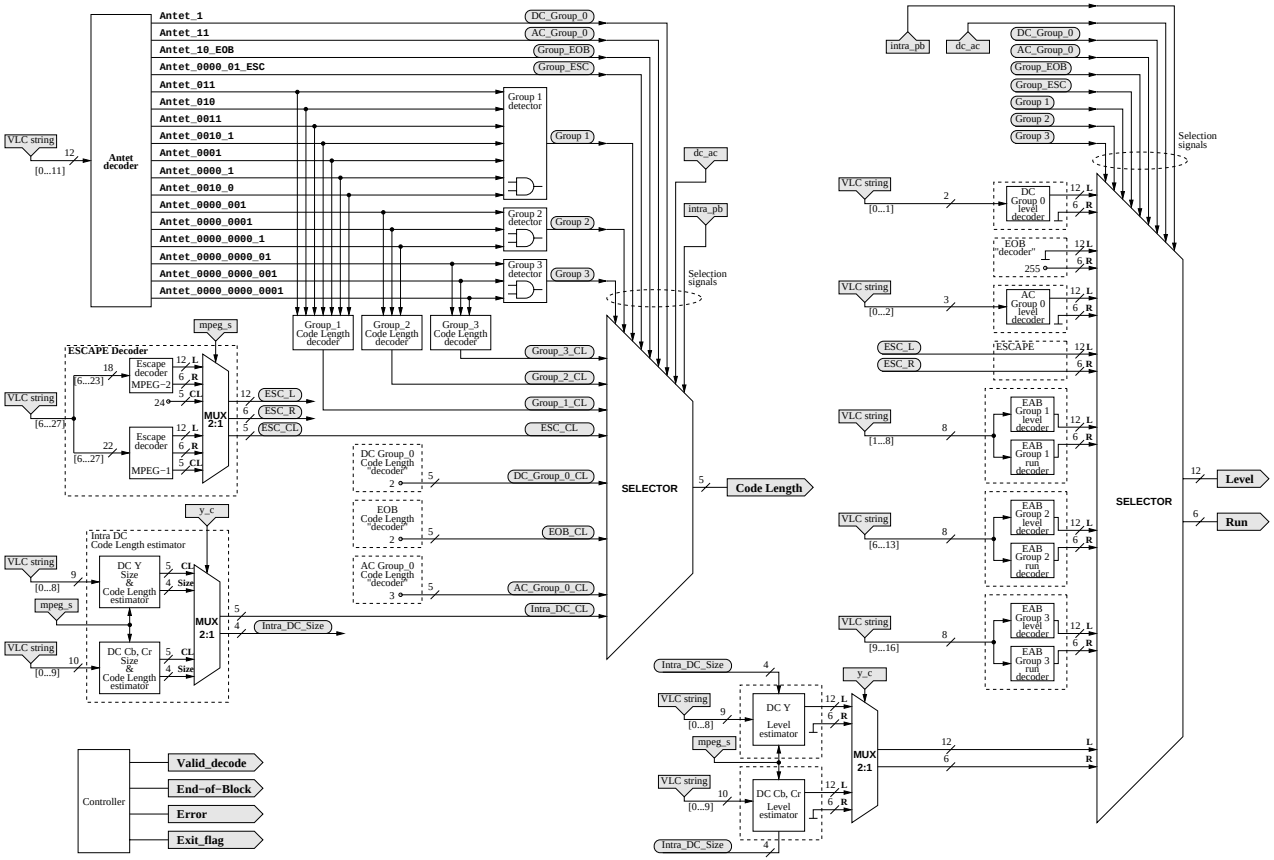


Fig. 9. The VLD implementation on FPGA.

6 8×8 IDCT

The functionality of the 8×8 IDCT can be implemented in both software and reconfigurable hardware. We will evaluate their performance subsequently.

6.1 8×8 IDCT implementation on standard TriMedia

In the current implementation of the 2-D IDCT on the standard TriMedia/CPU64 architecture, all computations are done with 16-bit values, and make intense use of SIMD-style operations. The 8×8 matrix is stored in sixteen 64-bit words, each containing a half row of four 16-bit elements. Therefore, four 16-bit elements can be processed in parallel by a single word-wide operation. Next to that, being a 5-issue slot VLIW processor, TriMedia/CPU64 can execute 5 such operations per clock cycle.

This strategy is used for both the horizontal and vertical IDCTs. First, eight 1-D IDCTs (two SIMD 1-D IDCTs) are computed using the modified 'Loeffler' algorithm [9]. Then, the transpose of the 8×8 matrix is performed by `TRANSPOSE` double-slot operations. Such a unit can generate the upper respectively lower two words of a transposed 4×4 matrix in one cycle. Therefore, the 8×8 matrix transpose is computed in eight basic operations. Finally, eight 1-D IDCTs (two SIMD 1-D IDCTs) are computed having the results generated by the transposition as inputs. Following the described procedure, a complete 2-D IDCT including all overheads (mostly composed of load and store operations) can be performed in 56 cycles [18].

6.2 8×8 IDCT implementation on extended TriMedia

As described in Section 4, a super-operation which can compute the 1-D IDCT on eight 16-bit values represented as two 64-bit words is available in extended TriMedia. The 1-D IDCT operation has a latency of 16, a recovery of 2, and can be issued on the slot pair 1+2. To calculate the 2-D IDCT, eight 1-D IDCT are firstly computed. Then, eight `TRANSPOSE` super-operations are scheduled on the slot pairs 1+2 or 3+4 to transpose the 8×8 matrix. Finally, eight 1-D IDCTs complete the 2-D IDCT. Before and after each 2-D IDCT, `LOAD` and `STORE` operations fetch the input operands from main memory into register file, and store the results back into memory, respectively.

In order to keep the pipeline full, back-to-back 1-D IDCT operation is needed. That is, a new 1-D IDCT instruction has to be issued every two cycles. Since true dependencies forbid issuing the last eight 1-D IDCTs of a 2-D IDCT so that to fulfill back-to-back requirement, the 2-D IDCTs are processed in chunks of two, in an interleaved fashion. A number of $2 \times 16 = 32$ registers are needed for this interleaved processing pattern. The code was manually scheduled. We found that the computational performance of 2-D IDCT exhibited a throughput of $1/32$ IDCT/cycle and a latency of 42 cycles [10].

7 Entropy decoder

The functionality of the entropy decoder can be implemented in both software and reconfigurable hardware. We will evaluate their performance subsequently.

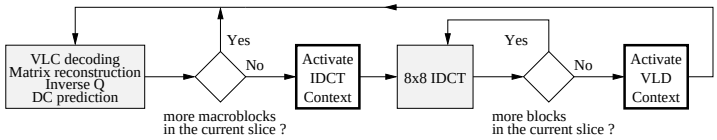


Fig. 10. The computing scenario of the macroblock parsing and pel reconstruction routine.

7.1 Entropy decoder implementation on standard TriMedia

The implementation of the entropy decoder in the standard TriMedia is a modified version of that proposed in [25]. The VLD has variable input-output rate, being implemented as a repeated table-lookup. Each lookup decodes a chunk of bits (8 bits at the first level lookup), and determines if a valid code was encountered. In case of a valid decode, a run-level pair is generated, or an escape or end-of-block flag is set. If a *miss* is detected, an offset into the VLC table and a chunk-size for a second-level lookup is generated. This process of signaling an incomplete decode and generating a new offset may be repeated three times. When a valid symbol has been encountered, it is stored into the 8×8 matrix at the location defined by the *run* value. After compiling the C code and scheduling procedure, we evaluated that a table lookup takes 21 cycles. Consequently, the entropy decoding of a single DCT coefficient can take between 21 and 63 cycles. The size of all lookup tables is 10 KB.

7.2 Entropy decoder implementation on extended TriMedia

The entropy decoder in the extended TriMedia benefits of reconfigurable hardware support. By employing software pipelining techniques, useful computations related to run-length decoding may be performed in the delay slots of the VLD operation. That is, the 8×8 empty matrix is succesively filled in with *level* values at the positions specified by *run* values. In this way, a symbol is processed completely in one (fixed latency) iteration. By simulation, we evaluated that a single DCT coefficient can be decoded in 11 cycles including all overheads.

8 Experimental results

In order to determine the potential impact on performance provided by the multiple-context reconfigurable core, we will consider a benchmark which consists of a macroblock parsing followed by pel reconstruction procedures. Therefore, we operate at MPEG slice level, i.e., the *data elements* on slice and above layers are assumed to be constant. The computing scenario is presented in Figure 10. First, a variable-length decoding of a macroblock (header and DCT coefficients extraction) is performed. Then, the 8×8 matrices are recreated, and inverse quantization, followed by DC coefficient prediction for intra-coded macroblocks are carried out. After all macroblocks in a slice have been decoded, a burst of 2-D IDCTs is launched in order to reconstruct the initial pels. During computation, the 1-D IDCT and VLD computing resources are activated by an `ACTIVATE_CONTEXT`, as needed.

All the contexts of the RFU are to be configured at application load time, i.e., a number of `SET_CONTEXT` instructions are scheduled on the top of the program code. A sample of the code using the instructions of the architectural extension is presented subsequently. As it can be observed, the `VLD` and `IDCT` exhibit the same execution pattern: two inputs and two outputs.

```
.alias    VLD                  EXEC_3   ; alias of the VLD instruction
.alias    IDCT                 EXEC_3   ; alias of the IDCT instruction
          SET_CONTEXT          VLD      ; load context VLD
          SET_CONTEXT          IDCT     ; load context IDCT
...
          ACTIVATE_CONTEXT VLD          ; configure VLD resource
...
          VLD Rx, Ry → Rz, Rw           ; execute VLD
...
          ACTIVATE_CONTEXT IDCT         ; configure IDCT resource
...
          IDCT Rx, Ry → Rz, Rw          ; execute IDCT
...
```

Therefore, our experiment includes two approaches: *pure software* and *FPGA-based*. As mentioned, a DCT coefficient is decoded in 21-63 cycles, and a 2-D IDCT can be computed in 56 cycles in the pure software approach. In the FPGA-based approach, a DCT coefficient is decoded in 11 cycles, and the 2-D IDCT is carried out with the throughput of 1/32 IDCT/cycle. Based on the published work in the field of multiple-context FPGAs [16], we make a conservative assumption and consider that the context switching penality is 10 cycles.

8.1 Pel reconstruction performance evaluation

A program which is MPEG-compliant has been written in C, compiled and scheduled with TriMedia development tools. The performance evaluation has been done assuming that, despite of the large lookup tables which are stored into memory, the standard TriMedia/CPU64 will never cope with a cache miss. In other words, we compare an 'ideal-cache" standard TriMedia with a multiple-context FPGA-augmented TriMedia.

Subsequently, we present the results according to two scenarios: *worst-case*[3] and *average-case*. In both cases we assumed that an average of 5 coefficients per block are decoded. In the worst-case scenario, we assumed that all DCT coefficients produce a *hit* on the first level lookup when the pure software implementation is used. In the same worst-case scenario, we also assumed that the overhead introduced by parsing the macroblock headers has the largest value (for example, the quantization value is assumed to be updated every macroblock). Since the worst-case scenario coresponds to long variable-length codes, it is statistically not relevant. Therefore, we evaluated the performances in a average-case scenario. In such scenario, we assumed that two of five DCT coefficients produce a *miss* at the first lookup. Also, we weighted the

[3] Considered from our point of view.

overhead introduced by parsing the macroblock header with the transmiting probability of different decoding parameters of the macroblock layer. The results are presented in Table 7. The numbers indicate the improvements we get for the number of cycles.

Table 7. Performance improvement of multiple-context FPGA-augmented TriMedia/CPU64 over 'ideal-cache" (standard) TriMedia/CPU64 for a macroblock parsing followed by pel reconstruction application.

		Worst-case scenario	*Average-case scenario*
Intra-coded macroblocks	prior to IDCT	15%	25%
	after IDCT	19%	29%
P-coded macroblocks	prior to IDCT	10%	21%
(1 block / macroblock)	after IDCT	14%	25%
P-coded macroblocks	prior to IDCT	13%	24%
(3 blocks / macroblock)	after IDCT	18%	27%
B-coded macroblocks	prior to IDCT	8%	17%
(1 block / macroblock)	after IDCT	12%	20%
B-coded macroblocks	prior to IDCT	11%	22%
(3 blocks / macroblock)	after IDCT	17%	25%

Finally, we proceeded to a global evaluation of the performance improvement. For an MPEG string with 10% intra-coded, 70% B-coded, and 20% P-coded macroblocks, the improvement for augmented TriMedia is $20 - 25\%$ in the average-case scenario.

9 Conclusions

We have proposed an architectural extension for TriMedia/CPU64 which encompasses a multiple-context FPGA-based reconfigurable functional unit and the associated instructions. On the augmented TriMedia/CPU64, we estimated a performance improvement of $20 - 25\%$ over a standard TriMedia/CPU64 for a macroblock parsing followed by a pel reconstruction application, at the expenses of three new instructions: `SET_CONTEXT`, `ACTIVATE_CONTEXT`, `EXECUTE`. As future work, we intend to consider the motion compensation and to evaluate the performance improvement for a complete MPEG decoder.

References

1. Razdan, R., Smith, M.D.: A High Performance Microarchitecture with Hardware-Programmable Functional Units. In: 27th Annual Intl. Symposium on Microarchitecture – MICRO-27, San Jose, California, (1994) 172–180.
2. Wittig, R.D., Chow, P.: OneChip: An FPGA Processor With Reconfigurable Logic. In: IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, (1996) 126–135.
3. Hauser, J.R., Wawrzynek, J.: Garp: A MIPS Processor with a Reconfigurable Coprocessor. In: IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, (1997) 12–21.

4. Kastrup, B., Bink, A., Hoogerbrugge, J.: ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator. In: IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, (1999) 92–100.
5. Mitchell, J.L., Pennebaker, W.B., Fogg, C.E., LeGall, D.J.: MPEG Video Compression Standard. Chapman & Hall, New York, New York (1996).
6. Sun, M.T.: Design of High-Throughput Entropy Codec. In: VLSI Implementations for Image Communications. Volume 2. Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1993) 345–364.
7. Rao, K.R., Yip, P.: Discrete Cosine Transform. Algorithms, Advantages, Applications. Academic Press, San Diego, California (1990).
8. Loeffler, C., Ligtenberg, A., Moschytz, G.S.: Practical Fast 1-D DCT Algorithms with 11 Multiplications. In: Intl. Conference on Acoustics, Speech, and Signal Processing (ICASSP '89), (1989) 988–991.
9. van Eijndhoven, J., Sijstermans, F.: Data Processing Device and method of Computing the Cosine Transform of a Matrix. PCT Patent No. WO 9948025 (1999).
10. Sima, M., Cotofana, S., van Eijndhoven, J.T., Vassiliadis, S., Vissers, K.: 8×8 IDCT Implementation on an FPGA-augmented TriMedia. In: IEEE Symposium on FPGAs for Custom Computing Machines, Rohnert Park, California, (2001).
11. Mukherjee, A., Ranganathan, N., Bassiouni, M.: Efficient VLSI Design for Data Transformation of Tree-Based Codes. IEEE Transactions on Circuits and Systems **38** (1991) 306–314.
12. Kinouchi, S., Sawada, A.: Variable Length Code Decoder. U.S. Patent No. 6,069,575 (2000).
13. Lei, S.M., Sun, M.T.: An Entropy Coding System for Digital HDTV Applications. IEEE Transactions on Circuits and Systems for Video Technology **1** (1991) 147–155.
14. Brown, S., Rose, J.: Architecture of FPGAs and CPLDs: A Tutorial. IEEE Transactions on Design and Test of Computers **13** (1996) 42–57.
15. DeHon, A., T. Knight, J., Tau, E., Bolotski, M., Eslick, I., Chen, D., Brown, J.: Dynamically Programmable Gate Array with Multiple Context. U.S. Patent No. 5,742,180 (1998).
16. Trimberger, S., Carberry, D., Johnson, A., Wong, J.: A Time-Multiplexed FPGA. In: IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, California, (1997) 22–28.
17. ***: ACEX 1K Programmable Logic Family. Altera Datasheet, San Jose, California (2000).
18. van Eijndhoven, J.T.J., Sijstermans, F.W., Vissers, K.A., Pol, E.J.D., Tromp, M.J.A., Struik, P., Bloks, R.H.J., van der Wolf, P., Pimentel, A.D., Vranken, H.P.E.: TriMedia CPU64 Architecture. In: Intl. Conference on Computer Design, Austin, Texas, (1999) 586–592.
19. Sima, M., Vassiliadis, S., Cotofana, S., van Eijndhoven, J.T., Vissers, K.: A Taxonomy of Custom Computing Machines. In: First PROGRESS Workshop on Embedded Systems, Utrecht, The Netherlands, (2000) 87–93.
20. Pol, E.J.D., Aarts, B.J.M., van Eijndhoven, J.T.J., Struik, P., Sijstermans, F.W., Tromp, M.J.A., van de Waerdt, J.W., van der Wolf, P.: TriMedia CPU64 Application Development Environment. In: Intl. Conference on Computer Design, Austin, Texas, (1999) 593–598.
21. van Eijndhoven, J.: 16-bit compliant software IDCT on TriMedia/CPU64. Internal Report, Philips Research Laboratories (1997).
22. ***: IEEE Standard Specifications for the Implementations of 8×8 Inverse Discrete Cosine Transform. IEEE Std 1180-1990 (1991).
23. Choi, S.B., Lee, M.H.: High Speed Pattern Matching for a Fast Huffman Decoder. IEEE Transactions on Consumer Electronics **41** (1995) 97–103.
24. Min, K.-y., Chong, J.-w.: A Memory-Efficient VLC decoder Architecture for MPEG-2 Application. In: IEEE Workshop on Signal Processing Systems, Lafayette, Louisiana, (2000) 43–49.
25. Pol, E.J.D.: VLD Performance on TriMedia/CPU64. Internal Report, Philips Research Laboratories (2000).

Several considerations about the latency of an RFU-configured computing resource are worth to be provided. Due to realization constraints, the RFU is likely to be located far away from the Register File (RF) in the floorplan of the TriMedia/CPU64. The immediate effect is that there will be large delays in transferring data between the RFU and RF, and the RFU will not benefit from bypassing capabilities of the RF [18]. Consequently, *read* and *write back* cycles have explicitely to be provided. In such circumstances, the minimum latency of an RFU-based computing resource includes at least 1 cycle for reading the input arguments from register file, the absolute minimum combinatorial delay Δ_{FPGA} on FPGA, and 1 cycle for writing back the results to the register file. Assuming that the FPGA clock frequency is equal with half of TriMedia clock frequency [10], the absolute minimum RFU latency is 4 TriMedia cycles. Since a call of an RFU is quite expensive, it would be a good idea to minimize the number of RFU calls, i.e., computing resources which can perform complex operations have to be configured on the RFU.

Constraints and freedoms in configuring a VLD computing resource (on FPGA ??):

- The *latency* of such computing resource should be known at compiling time. Therefore, no benefits from decoding preferentially the short (high probable) codewords can be achieved.
- The latency of such computing resource should be as small as possible, as the only way to speed up the decoding process. Pipelining is of no use here vezi articolele cu sistemele cu reacţie care nu se pretează la pipelining.
- There are 12 EABs (256×16 words) on an EP1K100 (???). Therefore, the prefix methodology and, consequently, partitioning the VLC tables should be performed according to this FPGA architectural caracteristic.

Generally speaking, a constant-output-rate VLD computes the codeword length by comparing the leading bits of the incoming bit stream against a small table. The decoder then sends the code length and the leading bits to other feed-forward circuitry for further decoding and immediately shifts the input by a number of bits equal with *code length*, to move to the new leading bits of the input bit stream for decoding the next codeword.

The critical path within the system is always the feedback path because other feed-forward paths can be pipelined. That is, the processing speed is limited by the feedback computation time: the time for comparing and selecting the codeword length plus the time for shifting the input [pag. 198, Lin & Messerschmitt, part. II]. The latency of computing the feedback value sets the decoding cycle time, and is thus inversely proportional to the decoding rate.

The performance metric is throughput, i.e., the net decoder information rate. This rate equals the number of bits or codewords decoded per cycle multiplied by the clock rate. There is a trade-off between these two terms; the more bits or codewords we try to decode in one cycle, the more complicated the PLA (look-up table !) will become and the slower the clock rate is. Pay attention! TriMedia has a fixed clock rate, the clock frequency is constraint, it is an input datum to the design of an MPEG decoder.

In cases where the number of codewords in the table is large, there are some bits that are common to the long VLC's, which we call *prefix*. By exploiting these common prefixes, the size of the LUT can be reduced. A number of schemes such as prefix precoding [Choi Lee], [Min Chong] and table partitioning [Cho Xanthopoulos...] have been presented ...

For FPGA-2002: advanced computation of the next code length. The selection of the proper result is performed simultaneously with the selection of the proper run and length of the *current* word. Also, in parallel, computing the run and length of the *previous* codeword is carried out.