Journal of VLSI Signal Processing 39, 195–212, 2005

IEEE-Compliant IDCT on FPGA-Augmented TriMedia

MIHAI SIMA
Department of Electrical and Computer Engineering, University of Victoria, P.O. Box 3055, Stn CSC, Victoria, B.C. V8W 3P6, Canada

SORIN COŢOFANĂ
Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

JOS T.J. VAN EIJNDHOVEN
Department of Information and Software Technologies, Philips Research Laboratories, Professor Holstlaan 4, 5656 AA Eindhoven, The Netherlands

STAMATIS VASSILIADIS
Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

KEES VISSERS
Electrical Engineering and Computer Sciences Department, University of California, 337 Cory Hall, Berkeley, California 94720-1774, USA

Received October 9, 2003; Revised October 9, 2003; Accepted January 21, 2004

Abstract. This paper presents a TriMedia processor extended with an IDCT reconfigurable design, and assesses the performance gain such an extension has when performing MPEG-2 decoding. We first propose the skeleton of an extension of the TriMedia architecture, which consists of a Field-Programmable Gate Array (FPGA)-based Reconfigurable Functional Unit (RFU), a Configuration Unit managing the reconfiguration of the RFU, and their associated instructions. Then, we address the computation of the 8×8 (2-D) IDCT on such extended TriMedia and propose a scheme to implement the 1-D IDCT operation on the RFU. When mapped on an ACEX EP1K100 FPGA from Altera, the proposed 1-D IDCT exhibits a latency of 16 and a recovery of 2 TriMedia@200 MHz cycles, and occupies 45% of the logic cells of the device. By configuring the 1-D IDCT on the RFU at application launch-time, the IEEE-compliant 2-D IDCT can be computed with the throughput of 1/32 IDCT/cycle. This figure translates to an improvement over the standard TriMedia of more than 40% in terms of computing time when 2-D IDCT is carried out in the framework of MPEG-2 decoding. Finally, the proposed reconfigurable IDCT is compared to a number of existing designs.

Keywords: configurable computing, VLIW processor, field-programmable gate array, inverse discrete cosine transform

1. Introduction

Inverse Discrete Cosine Transform (IDCT) constitutes an important operation of MPEG-related standards and has found wide applications in other fields (e.g., digital filtering) as well. Traditionally, IDCT has been implemented in hardware for Application-Specific Instruction Processors (ASIP), or in software in media-domain processors. In this paper, we propose a reconfigurable IDCT design for the 64-bit instance of TriMedia [1]. In order to assess the potential of the proposed design, we consider the TriMedia processor extended with an FPGA-based Reconfigurable Functional Unit (RFU). Using such RFU, we implement an 1-D IDCT operation and establish the gains in performance when computing a 2-D (8×8) IDCT.

Many algorithms have been proposed for efficient calculation of the IDCT. An 8×8 IDCT coded with a modified 'Loeffler' algorithm [2] can be scheduled in the standard instruction set of TriMedia in 56 cycles [1]. Since the standard TriMedia provides good support for transposition and matrix storage, we decided to provide RFU-hardware support only for an 1-D IDCT operation. When mapped on an ACEX EP1K100 FPGA, the 1-D IDCT computing unit exhibits a latency of 16, a recovery of 2 TriMedia@200 MHz cycles, and occupies 45% of the logic cells of the device. By configuring the 1-D IDCT unit on the RFU at application launch-time, the IEEE-compliant 2-D IDCT can be computed with the throughput of 1/32 IDCT/cycle. This figure translates to an improvement over the standard TriMedia of more than 40% in terms of computing time when 2-D IDCT is carried out in the framework of MPEG-2 decoding. Given the fact that the experimental TriMedia is a 5 issue-slot VLIW processor with 64-bit datapaths and a very rich multimedia-oriented instruction set [1], such an improvement within its target media processing domain [3] indicates that augmenting TriMedia with an FPGA shows clear benefit for doing 2-D IDCT.

The paper is organized as follows. For background purpose, we briefly present the most important issues related to IDCT theory and architecture of the reconfigurable core in Section 2. Section 3 describes the proposed architectural extension of TriMedia. The current implementation of the 8×8 IDCT on standard TriMedia, implementation issues of the 1-D IDCT computing resource on FPGA, the execution scenario of the 2-D IDCT on the extended architecture, as well as experimental results are presented in Section 4. Section 5 completes the paper with some conclusions and closing remarks.

2. Background

In this section, we briefly present a theoretical background of IDCT. We also review the architecture of the FPGA we used as an experimental reconfigurable core.

Inverse Discrete Cosine Transform. The transformation for an N point 1-D IDCT is defined by [4]:

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

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]:

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

A standard strategy to compute the 2-D IDCT is the 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 [6]. It has to be mentioned that the Loeffler algorithm has an intrinsic gain of $2\sqrt{2}$; thus, after the vertical and horizontal 1-D IDCTs have been computed, a factor of $2\sqrt{2} \times 2\sqrt{2} = 8$ has to be compensated out. This can be easily achieved by right-shifting by three positions.

To fulfill the IEEE numerical accuracy requirements for IDCT in MPEG applications [7], van Eijndhoven and Sijstermans proposed a slightly different version of the Loeffler algorithm in which the $\sqrt{2}$ factors are moved around [2]. In our experiment, we will use this modified algorithm (see Fig. 1), which has also an intrinsic gain of $2\sqrt{2}$.

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 Fig. 2.

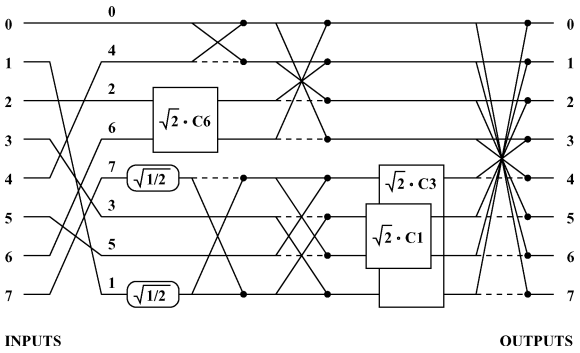

Figure 1. The modified 'Loeffler' algorithm.

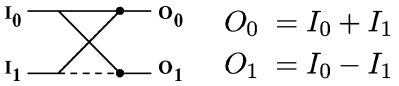

Figure 2. The butterfly—[6].

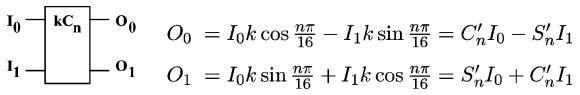

Figure 3. The rotator—[6].

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 Fig. 3.

Although an implementation of such a rotator with three multiplications and three additions is possible (Fig. 4(a) and (b)), we used the direct implementation of the rotator with four multiplications and two additions (Fig. 4(c)), because it shortens critical path and improves numerical accuracy. Indeed, there are three operations (two additions and a multiplication) on the critical path of the implementations with three multipliers, while the critical path of the implementation with four multipliers contains only two operations (a multiplication and an addition). Also, the initial addition involved by the three-multiplier implementations may lead to an overflow when fixed-point arithmetic is carried out.

The FPGA Architecture. Field-Programmable Gate Arrays (FPGA) [8] 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 information stored into the configuration memory defines the function performed by the raw hardware. In this paper, we assume that the architecture of the raw hardware is identical to that of an ACEX 1K device from Altera [9]. Both ACEX 1K and TriMedia families are of about the same generation and manufactured in the same TSMC technological process. This choice allows us to make realistic assumptions regarding the performance of FPGA-mapped logic versus a hypotethical hardwired counterpart.

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 a RAM block with 8 inputs and 16 outputs, and a rich interconnection network. The logic capacity of the ACEX 1K family ranges from 576 logic cells for EP1K10 device to 4992 logic cells for EP1K100 device. The absolute maximum rating for the frequency of synchronous designs mapped on these FPGAs 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 [9].

The next section will introduce the architectural extension for the experimental 64-bit TriMedia instance, which is also referred to as TriMedia–CPU64.

3. TriMedia Architectural Extension

TriMedia–CPU64 is a processor model, whose architecture features a rich instruction set optimized for

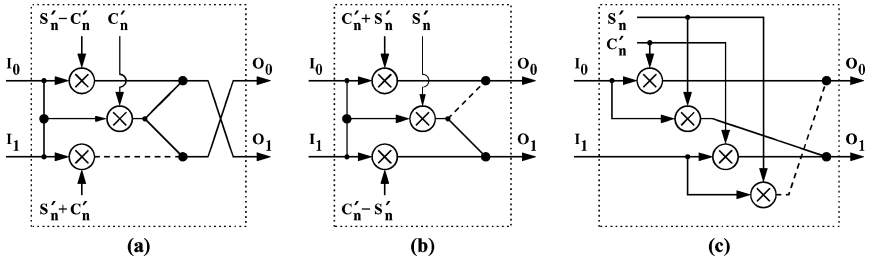

Figure 4. Three possible implementations of the rotator.

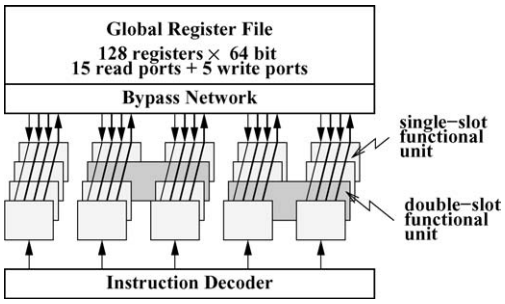

Figure 5. TriMedia–CPU64 organization—[1].

media processing. Specifically, it is a 5 issue-slot VLIW engine, launching a long instruction every clock cycle [1]. It has a uniform 64-bit wordsize through all functional units, the register file, load/store units, internal and external buses. Each of the five operations in a single instruction can in principle read in two register arguments and write back one register result. In addition, each operation can be guarded with the least-significant bit of a fourth register to allow for conditional execution without branch penalty. With the exception of floating point divide and square root unit, all functional units have a recovery of 1, while their latency ranges from 1 to 4 (*latency* is the number of clock cycles between the issue of an operation and availability of its results, while *recovery* is defined as the minimum number of clock cycles between the issue of successive operations). The TriMedia core is assumed to support multiple-slot operations, or super-operations [10]. Such a super-operation occupies two or more adjacent slots in the VLIW instruction, and maps to a wider functional unit. This way, operations with more than two arguments and one result are possible. The architecture also supports subword (SIMD-style) parallelism on byte, half-word, or word entities. The current organization of the TriMedia-CPU64 is presented in Fig. 5.

In this paper, we propose to augment the TriMedia–CPU64 processor with a Reconfigurable Functional Unit (RFU) consisting of an FPGA core and its associated Controller, and a Configuration Unit (CU) managing the reconfiguration of the FPGA. Both the RFU and CU are embedded into TriMedia as any other hardwired functional units, i.e., they receive instructions from the instruction decoder, read their input arguments from and write the computed values back to the register file, as depicted in Fig. 6. In this way, only minimal modifications of the basic architecture, and also of the associated compiler and scheduler are required.

In order to use the RFU based on the MOLEN architectural and programming model [11], the user is provided a kernel of new instructions: `SET`, and `EXECUTE`. This kernel constitutes the extension of the TriMedia instruction set architecture we propose. Loading configuration information into the FPGA configuration memory is performed by the Configuration Unit under the command of a `SET` instruction, while the `EXECUTE` instruction launches the operation performed by the FPGA-mapped computing unit [12]. With these new instructions, the user is given the freedom to define and use any computing unit subject to the FPGA size and TriMedia organization.

Uploading configuration information to the CU is performed under the command of a double-slot instruction issued on Slot pair 1 + 2:

$$\text{SET} \quad \text{Rs1, Rs2, Rs3} \rightarrow \text{Rd}$$

where the registers Rs1, Rs2, and Rs3 contain 192 bits of configuration information. If the instruction completes successfully, then the register Rd contains 0, otherwise it contains an error code. For the ACEX 1K family, which in fact we assume in our subsequent experiment, the average latency for loading new FPGA configuration information from off-chip

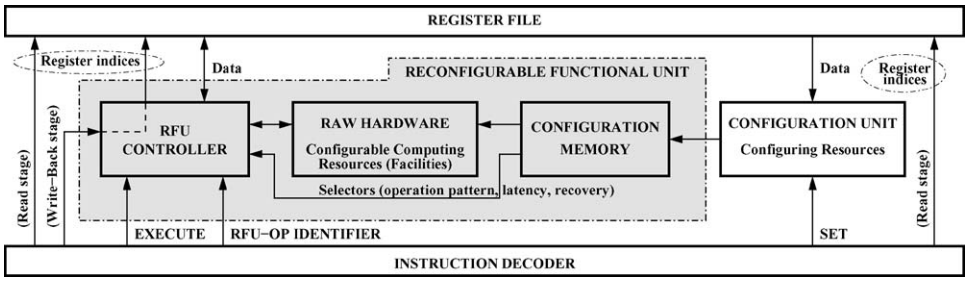

Figure 6. The architectural extension of TriMedia–CPU64 VLIW core.

is about 50 ns/byte, that is 10 cycles/byte. Since the SET instruction places 192 bits = 24 bytes on the CU at a time, it has a latency of 240 cycles. For an EP1K100 device, which has a configuration file of 1,337,000 bit [9], 6,964 SET instructions or 6,964 × 240 = 1,671,360 cycles are needed to reconfigure the array.

Conceptually speaking, computing units of user-definable computing pattern,[1] latency, recovery, and slot-assignment[2] can be configured on RFU. Thus, the RFU can act as five independent single-slot functional units each of them executing a different custom operation, a mixture of single- and multiple-slot functional units, or even a five-slot functional unit. In all these situations, the RFU may receive EXECUTE instructions issued on any of the five TriMedia slots, and use all 10 read and 5 write ports of the register file per call.

In connection to the FPGA-augmented TriMedia implementation, we would like to note that the flexibility in defining slot-width and slot-assignments for RFU-mapped operations determines the implementation cost. For example, assuming the maximum freedom degree in defining slot-assignments for RFU operations, a separate RFU controller has to be placed on each issue slot. Moreover, the TriMedia instruction decoder has to be able to decode EXECUTE instructions on each of the five issue slots. In addition, we also note that although the maximum flexibility in defining RFU-based operations may be of theoretical value, it is not of practical relevance in the context of our current investigations. As one can notice in the next sections, the RFU-mapped IDCT is a double-slot operation.

For this reason, in this paper we consider only a particular instance of FPGA-augmented TriMedia, in which only a single-slot instruction on Slot 1 and a double-slot instruction on Slot pair 1 + 2 can be issued to the RFU. Our choice does not require additional development efforts. Since only single- and double-slot operations are currently supported by the compiler and scheduler for the time being, there is no need to modify the TriMedia toolchain.

For each of the single-slot, and double-slot RFU instructions, a separate operation code is allocated: EXECUTE_1, and EXECUTE_2, respectively. In both cases, the standard TriMedia–CPU64 instruction format is preserved: the opcode is a 9-bit field, and each and every source or destination registers is specified by a 7-bit field. Up to two inputs and one output, and four inputs and two outputs can be specified by the single-, and double-slot instructions, respectively:

EXECUTE_1 Rs1, Rs2 → Rd1

EXECUTE_2 Rs1, Rs2, Rs3, Rs4 → Rd1, Rd2

The EXECUTE instructions are generic, since their semantics can be redefined. By reconfiguring the raw hardware, followed by issuing an EXECUTE instruction, any new user-defined operation subject to FPGA size and TriMedia organization can be launched into execution, while only a single entry in the opcode space is needed to encode the EXECUTE instruction. Since all the fields in the EXECUTE_1 instruction format but the opcode field encode the input and output registers, there are no provisions for additional encoding. Thus, only a single operation per RFU configuration can be encoded within EXECUTE_1. That is, if a different single-slot operation is to be launched, then a reconfiguration of the raw hardware must be carried out beforehand. However, as we describe subsequently, more operations per RFU configuration can be encoded within a multiple-slot EXECUTE instruction. This may reduce the number of reconfigurations when a large FPGA is available.

In the standard TriMedia–CPU64, only one of the *opcode* fields in a multiple-slot instruction defines the operation, all the others being set NOPs (Table 1). By using these unused fields as an argument for the RFU OPCODE (Table 2), a large number of RFU operations can be encoded per configuration, while only a single entry for the EXECUTE instruction needs to be allocated in the opcode space. Assuming a double-slot operation, for example, the 9-bit additional opcode (which is subsequently referred to as an RFU-OP-IDENTIFIER or simple RFU-OP-ID) can specify 512 different operations.

We would like to mention that the two parts of the double-slot operation are decoded separately, and only when the first part specifies an EXECUTE_2 opcode,

Table 1. The VLIW double-slot operation instruction format.

Slot 1		Slot 2		3, 4, 5
OPCODE	src. & dest.	NOP	src. & dest.	...

Table 2. The RFU double-slot instruction format.

Slot 1		Slot 2		3, 4, 5
RFU OPCODE	src. & dest.	*RFU-OP ID*	src. & dest.	...

the second opcode is interpreted as an RFU-OP-IDENTIFIER, and thus decoded locally at the RFU by the RFU controller. This way, an RFU super-operation does not create pressure on the instruction decoder, neatly fits in the existing instruction format, fits the existing connectivity structure to the register file, and hence requires very little hardware overhead.

In connection to the EXECUTE instructions, we would like to emphasize that in addition to semantics, their number of operands, latency, and recovery are user-definable, too (the slot-width is defined implicitly by the particular EXECUTE_1 or EXECUTE_2 opcode). Thus, it is the responsibility of the programmer to augment the Machine Description File with appropriate information [13]. At the machine implementation level, these parameters are set by means of *Selectors*, which become part of the RFU configuration, as presented in Fig. 6. A different {*number of operands*, *latency*, *recovery*} set can be defined for each RFU-OP-ID. With such mechanism, an EXECUTE instruction is trully generic, and the programmer is able to adjust its behavior as needed.

In a similar way, the user can define as many RFU-related instructions as he/she wants. In the subsequent experiment, we will define a single RFU-related operation which computes an 1-D IDCT on eight points. We will present implementation issues of the 1-D IDCT computing facility on the FPGA, and will evaluate the performance when computing an 8×8 IDCT.

4. Experimental Results

In order to determine the potential impact on performance provided by the reconfigurable core, the 1-D IDCT is configured on the RFU at application launch-time. That is, the SET instructions are scheduled on the top of the program. As mentioned, we use an ACEX 1K FPGA from Altera as experimental platform for the reconfigurable core.

In the 2-D IDCT implementation on standard 5-issue slot TriMedia, all computations are done with 16-bit values, and make intense use of four-way 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, since the host is a 5-issue slot VLIW processor, five such operations can be executed per clock cycle.

To calculate the 2-D IDCT, eight 1-D IDCTs are first computed using the modified 'Loeffler' algorithm [2]. By using the four-way SIMD operations, four IDCTs are effectively computed concurrently. That is, the eight 1-D IDCTs are calculated as two four-way SIMD 1-D IDCTs. Then, the transpose of the 8×8 matrix is performed by a transpose unit which covers a double issue slot. A TRANSPOSE double-slot operation can generate either the top or bottom transpose half of a transposed 4×4 matrix in one cycle. Therefore, the transpose of an 8×8 matrix is computed in eight basic operations. Finally, eight 1-D IDCTs (two SIMD 1-D IDCTs) are computed with the results generated by the transposition. Following the described procedure, a complete 2-D IDCT including the load and store operations can be performed in 56 cycles [1].

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 issue-slot occupancy. Consequently, in our implementation on the extended TriMedia, we configure only an 1-D IDCT double-slot computing resource on the RFU. By launching an 1-D IDCT double-slot operation having two 64-bit inputs and two 64-bit outputs, an 1-D IDCT is computed on eight 16-bit values. To calculate the 2-D IDCT, eight 1-D IDCT are firstly computed. Then a transpose is performed on the 8×8 data matrix using TriMedia native TRANSPOSE double-slot operations. Finally, eight 1-D IDCT are again computed. This execution scenario is presented in Fig. 7.

Let us assume that a horizontal packing of the data is needed at the output of the 8×8 IDCT, i.e., four elements in the same matrix line are to be stored into a (64-bit) double word. If the input data is also horizontally packed, then two transposition stages are needed (Fig. 8—left), otherwise, only the middle transposition stage is required (Fig. 8—right). In connection to the MPEG decoding task [5], we notice that the front transposition stage can be bypassed if the appropriate *zig-zag* scan ordering or its transposed version is used

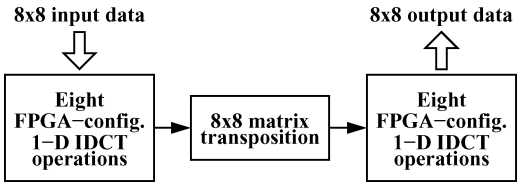

Figure 7. The computing scenario of 8×8 IDCT on the extended TriMedia.

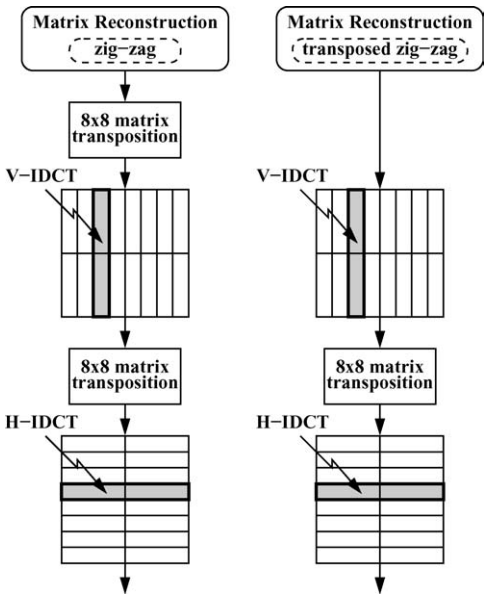

Figure 8. Bypassing the first transposition stage.

for 8×8 matrix reconstruction. Consequently, the performance evaluation of the 8×8 IDCT takes into consideration only eight 1-D IDCTs, a transposition stage, and again eight 1-D IDCTs.

A pipelined FPGA implementation of 1-D IDCT having a recovery of 1 implies that the FPGA clock frequency is equal with the TriMedia clock frequency. Nowadays, the upper limit of the clock frequency in TriMedia family is around 300 MHz, while the maximum clock frequency for ACEX 1K FPGA family 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, which translates into an FPGA cycle time to TriMedia cycle time ratio of 2. Our assumption does not violate the general accepted performance ratio of FPGA-mapped logic versus hardwired logic—see for example [14]. Considering a TriMedia running at 200 MHz, the pipelined implementation of 1-D IDCT will work with a clock frequency of 100 MHz.

Implementation Issues of the 1-D IDCT. All the operations required to compute 1-D IDCT are implemented using 16-bit fixed-point arithmetic. Referring again to Section 2, and to Figs. 1, 3, and 4, since the computation of 1-D IDCT requires 14 multiplications, an efficient implementation of each multiplication is of crucial importance. For all multiplications, the multiplicand is a 16-bit signed number represented in 2's complement notation, while the multiplier is a positive constant of 15 bits or less. As proved in [15], these word lengths in connection with fixed-point arithmetic and proper rounding are sufficient to fulfill the IEEE numerical accuracy requirements for IDCT in MPEG applications.

A general multiplication scheme for which both multiplicand and multiplier 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 redundant, 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 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 an FPGA family may not be appropriate for a different family.

In Table 3 we present the performances of several reduction modules for ACEX 1K. All the experiments correspond to synchronous designs, i.e., both inputs and outputs are registered. The figures have been obtained by compiling VHDL source code with Leonardo Spectrum™ from Exemplar, followed by placement and routing with MAX+PLUS II™ from Altera. Since a reliable broadcast of the clock signal on an ACEX 1K chip is not guaranteed for a frequency above 180 MHz [9], we will proceed to a conservative approach and consider the italics-typed figures as being too optimistic, although they have been generated by software tools. Consequently, we will operate such "high-frequency" reduction modules at 180 MHz or below. The following settings of the software tools have been used: (1) Leonardo-Spectrum™: *Lock LCELLs: NO*, *Map Cascades: YES*, *Extended Optimization Effort*, *Optimize for Delay*, *Hierarchy: Flatten*, *Add I/O Pads: NO*; (2) MaxPlus-II: *WYSIWYG*, *Optimize* = 10 *(Speed)*; (3) MaxPlus-II: *FAST*, *Optimize* = 10 *(Speed)*.

In order to implement an IDCT at 100 MHz, reduction modules which can run at 100 MHz or

Table 3. Performances of several reduction modules for ACEX 1 K Speed Grade-1.

Reduction module		Performance f_{max} (MHz)		
		Leonardo-Spectrum (1)	MaxPlus-II WYSIWYG (2)	MaxPlus-II FAST (3)
Two-operand	16-bit adder	136	140	140
Three-operand		104	107	117
Four-operand		104	103	109
Five-operand		84	81	81
Six-operand		84	76	76
Two-operand	24-bit adder	112	114	114
Three-operand		89	94	94
Four-operand		89	86	90
Two-operand	28-bit adder	102	103	103
Three-operand		83	85	83
Four-operand		83	77	81
Two-operand	30-bit adder	98	102	102
Three-operand		88	93	91
Five-operand	3-bit adder	108	147	138
Six-operand		108	131	121
Seven-operand		108	128	116
Five-operand	4-bit adder	105	126	113
Six-operand		105	126	107
Seven-operand		105	111	114
Five-operand	6-bit adder	101	113	107
Six-operand		101	97	105
Seven-operand		101	94	97
Three inputs	Dadda population counter	*231*	*250*	*250*
Four inputs		*228*	*250*	*250*
Five inputs		155	175	169
Six inputs		155	*188*	*188*

more should be considered. Subsequently, we present the reduction steps for all multiplications. In order to implement 16-bit fixed-point arithmetic, both the multiplicand and multiplier have been properly scaled so that values remain representable with 16 bits and 15 bits, respectively, while preserving the highest possible precision [15]. Also, only the most significant 16 bits of the extended 31-bit product are to be stored. It is worth mentioning that only 30 out of 31 bits of the product have to be computed. As depicted in Fig. 9(a), the 31st (most significant) bit (labeled '30') is derived from the sign-bit of the multiplicand, 'S', while the carry from position '29' to '30' is discarded.

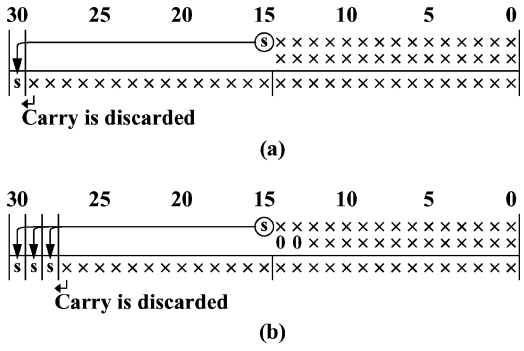

Figure 9. Sign-extension for: (a) 15-bit multiplier; and (b) 13-bit multiplier.

Recursively, assuming that the multiplier magnitude is small enough so that the multiplier can be represented with only n bits, where $15 \geq n \geq 1$, then only $15+n$ out of 31 bits of the extended product have to be computed. The most significant $15-n+1$ bits of the product are derived from the sign-bit of the multiplicand, while the carry from position $15+n-1$ to position 15 + n is discarded. Fig. 9(b) presents an example for $n = 13$.

In addition to the solution we described in [16], we implemented a *Rounding-To-Nearest* (rtn) scheme [17] at the end of each multiplication. Assuming that the extended 31-bit product is $p_{15}, \ldots, p_1, p_0, p_{-1}, p_{-2}, \ldots, p_{-15}$, where $p_{-1}, p_{-2}, \ldots, p_{-15}$ are the bits to be discarded, the *rounding-to-nearest* is performed by adding the bit p_{-1} to the 16-bit $(p_{15}, \ldots, p_1, p_0)$ unrounded product, as it is depicted in Fig. 10 ('S' represents the sign-bit, and 'R' specifies the rounding-bit). The generated rounding function is depicted in Fig. 11, where x is the value to be rounded, while rtn(x) is the rounded value nearest to x.

In connection to the rounding stage, several comments are worth to be provided. First, in order to obtain (16-bit) zero when rounding the (31-bit) highest negative numbers in the range `7fff 8000`...`7fff ffff`, carry propagation over all 16 bits of the rounded product is needed. That is, the sign bit has to be involved in computation during the rounding stage. This imposes a significant overhead as the final rounding is always on the critical path of the multiplication. Second, we remind that the 16-bit fixed-point arithmetic and *rounding-to-nearest* is sufficient to fulfill the IEEE numerical accuracy requirements for IDCT in MPEG applications. This was really confirmed by performing the IEEE accuracy validation [15]. This means that when rounding the (31-bit) highest positive numbers in the range `3fff 8000` ... `3fff ffff`, an overflow is never encountered. For this reason, saturating arithmetic is not needed and, therefore, has not been implemented. Finally, following the procedure described in [17], we estimate that the normalized magnitude of the upward *bias* introduced by the *rounding-to-nearest* scheme is:

$$\text{bias} = \frac{0.5}{2^{15-\text{atzlsb}}}$$

where *atzlsb* (*all-time zero least significant bits*) is the number of least significant $p_{-1}, p_{-2}, \ldots, p_{-15}$ bits of the product which are zero all the time. The worst case is encountered for multiplication by S'_1, where $atzlsb = 3$ (as we show later on). Therefore,

$$\text{bias} = \frac{0.5}{2^{12}} = \frac{0.5}{4096} = 0.00012207$$

which means that $4096 \times 2 = 8192$ rounding steps are needed to affect the precision of the 16-bit rounded product by only 1 bit. Fortunately, the number of the operations including a rounding step that are needed to reconstruct a pixel (both 2-D IDCT and motion compensation are considered) is far less that 8196 (is of the order of 20 or so). Consequently, the bias introduced by the *rounding-to-nearest* scheme does not affect the result and, therefore, is out of concern.

The partial product matrix and the selected reduction modules and steps for multiplication by the constant $C'_0 = 5a82\,\text{h}$ are presented in Fig. 12 (the Roman numerals indicate the reduction steps). First, the partial product matrix is built. Then, reductions on the modules specified by the shaded areas are carried out. The

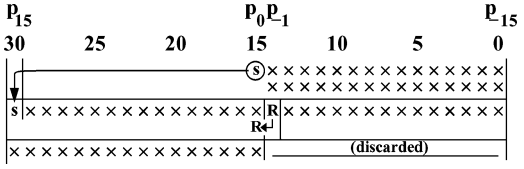

Figure 10. Rounding-To-Nearest implementation.

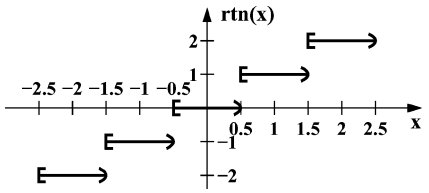

Figure 11. Rounding of a 2's-complement value to the nearest number.

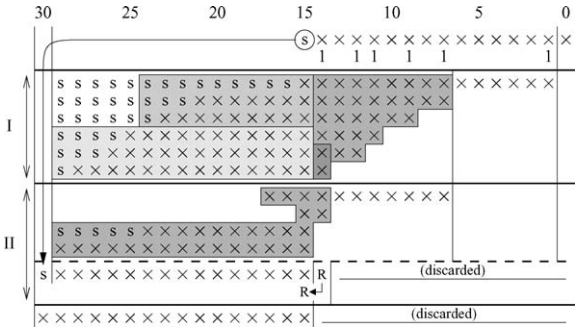

Figure 12. The partial product matrix and the selected reduction steps for multiplication by the constant C'_0.

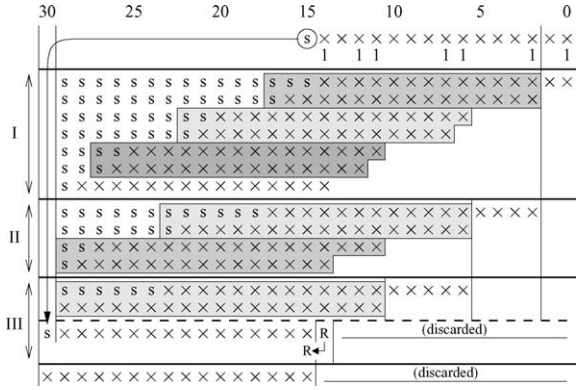

Figure 13. The partial product matrix and the selected reduction steps for multiplication by the constant C_1'.

first stage generates four binary numbers of different lengths result, which are reduced to one row in the second stage. Therefore, a multiplication by the constant C_0' including rounding is performed in two pipeline stages.

The partial product matrix and the selected reduction modules and steps for multiplication by the constant $C_1' = 58c5\,\mathrm{h}$ are presented in Fig. 13. The reduction is performed in a horizontal way, two lines at a stage. Therefore, a multiplication by the constant C_1' is performed in three stages. The multiplication by the constant C_1' proved too difficult to be implemented in two stages only.

The partial product matrix and the selected reduction modules and steps for multiplication by the constants $S_1' = 11a8\,\mathrm{h}$, $C_3' = 4b42\,\mathrm{h}$, $S_3' = 3249\,\mathrm{h}$, $C_6' = 22a3\,\mathrm{h}$, and $S_6' = 539f\,\mathrm{h}$ are presented in Figs. 14–18, respectively. Concerning the multiplication by constant S_6', some comments are worth to be provided. In order to reduce the number of '1' in the multiplier S_6' and,

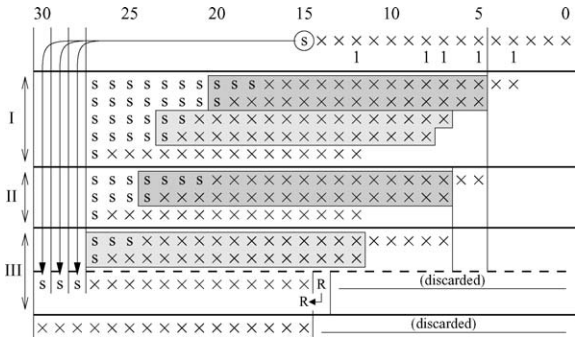

Figure 14. The partial product matrix and the selected reduction steps for multiplication by the constant S_1'.

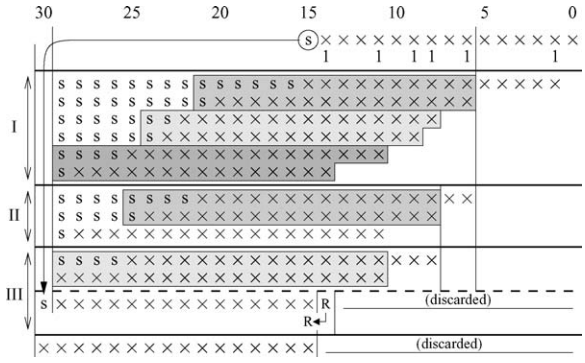

Figure 15. The partial product matrix and the selected reduction steps for multiplication by the constant C_3'.

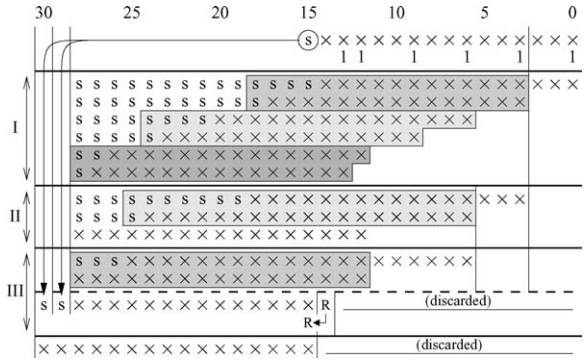

Figure 16. The partial product matrix and the selected reduction steps for multiplication by the constant S_3'.

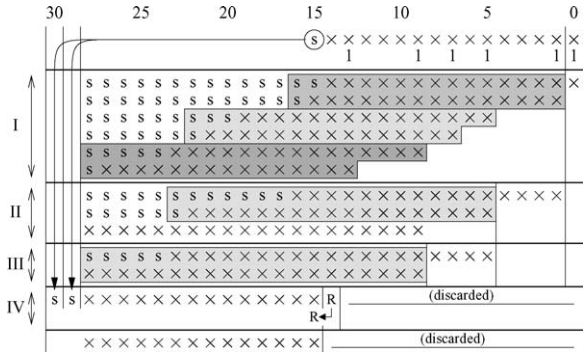

Figure 17. The partial product matrix and the selected reduction steps for multiplication by the constant C_6'.

consequently, the number of rows in the corresponding partial product matrix, the Booth's recoding [17] has been applied. That is, the multiplier S_6' is rewritten as $S_6' = 5420\,\mathrm{h} - 0081\,\mathrm{h}$, and the rows in the partial product matrix corresponding to 0081 h are subtracted rather than added.

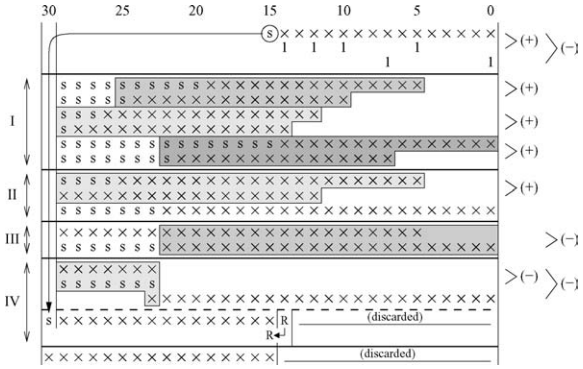

Figure 18. The partial product matrix and the selected reduction steps for multiplication by the constant S'_6.

We would like to note that the critical path of the 1-D IDCT is located on the bottom half of the modified 'Loeffler' algorithm (Fig. 1). Once the multiplication by constant C'_1 is performed in three stages, there is no gain in performance to implement the other three multiplications by constants S'_1, C'_3, S'_3 in less than three stages. Therefore, the multiplications by the constants S'_1, C'_3, S'_3 are implemented in three stages also, even though they may allow for an efficient (timing) implementation in two stages, too. The same timing-relaxed implementation strategy is used for multiplications by the constants C'_6 and S'_6, since they both are not located on the critical path.

The sketch of the 1-D IDCT pipeline is depicted in Fig. 19 (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 the additions for the butterflies between the multipliers by the constant C'_0 and the rotators $\sqrt{2}C_1$ and $\sqrt{2}C_3$.
- three FPGA cycles for computing the multiplication by the constant C'_1;
- one FPGA cycle for computing all the additions required by the rotators $\sqrt{2}C_1$ and $\sqrt{2}C_3$, and the final butterflies;
- 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 determined that 1-D IDCT uses 45% of the logic elements and 257 I/O pins of an ACEX EP1K100 device.

2-D IDCT on Extended TriMedia. As mentioned, an 1-D IDCT with a latency of 16 and a recovery of 2 is configured on the RFU at application launch-time. We assigned the IDCT operation to the slot pair $1+2$. After eight 1-D IDCTs, eight `TRANSPOSE` super-operations are scheduled on the slot pairs $1+2$ or $3+4$ to compute the transpose of the 8×8 matrix. Then, 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. The scheduled code and the performance figures are presented in Fig. 20.

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 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

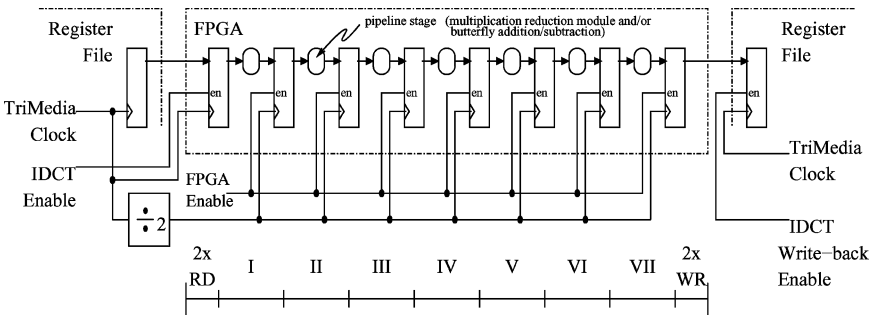

Figure 19. The 1-D IDCT pipeline.

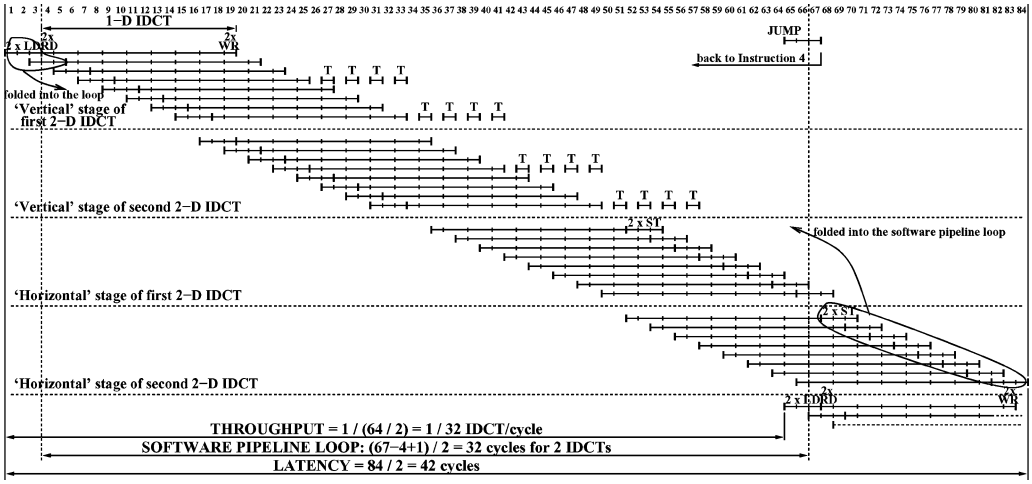

Figure 20. Schedule result for a 1-D IDCT having the latency of 16 and recovery of 2 (LD stands for `LOAD`, RD for read, WR for write, ST for `STORE`, and T for `TRANSPOSE`).

needed for this processing pattern. This 2-D IDCT implementation exhibits a throughput of 1/32 IDCT/cycle and a latency of 84 cycles for two IDCTs (that is, an average of 42 cycles/IDCT). It is worth mentioning that the machine is well balanced, none of the 5-slot VLIW instructions being fully occupied with operations:

- Two `LOAD` or two `STORE` operations are issued every other clock cycle on slots 4 and 5; thus the slots 4 and 5 are only 50% occupied.
- `IDCT` super-operations are issued on slots $1+2$ every other clock cycle, which translates to a 50% usage of the slots 1 and 2.
- The *transpose* super-operations are also issued on every other clock cycle, and the issuing slots can be either $1 + 2$ or $3 + 4$. Since there are only eight transpositions per 2-D IDCT, the overall slot occupancy percentage does not increase significantly above 50%.

In this way, there are plenty of free slots which can be utilized for other purposes, e.g., for implementing the post-IDCT rounding and saturation required by MPEG standard [5], or even a 2-D IDCT in the standard instruction set. Consequently, the announced figures represent the lower bound of the performance improvement which can be achieved on extended TriMedia.

In connection to the scheduled code presented in Fig. 20, we would like to mention that cycling over Instructions $1 \div 84$ is needed to launch the computation of the next two 2-D IDCTs. The immediate effect is that there is an overhead associated to firing-up and flushing the reconfigurable-hardware (1-D IDCT) pipeline. Thus, the throughput of 1/32 IDCT/cycle corresponds to the ideal scenario of a loop which is unrolled an infinite number of times.

In order to have a realistic scenario, two techniques can be employed: (1) finite loop unrolling, and (2) *software pipelining*. Both techniques will be analysed subsequently and performance figures will be provided. Concerning the second technique, we have to mention that, for the time being, the TriMedia scheduler uses the decision tree as a scheduling unit [18]. Thus, all operations return the results in the same decision tree that they are launched, even though the TriMedia architecture does not forbid the contrary. This is the major limiting factor in generating deep software pipelined loops containing long-latency operations. However, the code containing RFU operations is very simple; thus, programming in assembly is indeed a feasible solution despite of the fact that the host is a complex VLIW processor.

In Fig. 20, we also present the edges of the software pipeline loop (Instructions 4 and 67), as well as the corresponding `JUMP` operation which cycles over the loop. To employ loop pipelining, the first 4 `LOAD` operations and the last 16 `STORE` operations should be folded into the loop. Thus, the overhead associated to firing-up and flushing the software pipeline (i.e., the *prologue* and the *epilogue*) consists of these 4 `LOAD`, and 16 `STORE` operations, respectively, issued in pairs every other cycle. Thus, the total overhead is 20 cycles.

Table 4. MPEG-2 statistics for several conformance bit-strings.

Scene (MPEG-conformant bit-string)	Coded blocks/slice (average number)		
	Frame type		
	I	P	B
bat_327_334	–	257	234
popplen	264	80	38
sarnoff2	270	171	61
tennis	264	167	71
ti1cheer	264	155	88

In order to assess the implications of the loop prologue and epilogue in a real case, we have focused on the average number of coded blocks per slice for a number of MPEG-conformance bit-strings (Table 4). If all the blocks in an MPEG slice are first reconstructed and only then transformed as a single batch, then the lowest average batch size is 38 blocks/slice (B frames in the popplen scene). This figure translates in an average penality associated to the prologue and epilogue of the software pipeline loop of $20/38 \approx 0.54$ cycles/block. Since this overhead represents less than 2% of the 32 cycle/block throughput, it can be neglected.

In Table 5, we present performance figures for two loop organizations (linear and software pipelined), several computing scenarios (FPGA-based 2-D IDCTs are processed in chunks of two, FPGA-based 2-D IDCTs are blended with a single 2-D IDCT computed in software, vertical 1-D IDCTs and transpositions are computed first for all matrices of the testbench, and only then all horizontal 1-D IDCTs are carried out), and several degrees of loop unrolling. Since the IDCT rounding and saturation as specified by MPEG standard [5] may be subject to optimization at a complete MPEG decoder level, we will also present the experimental figures for three cases: IDCT rounding and saturation is performed in FPGA as an additional (the eighth) hardware pipeline stage, in software in the standard TriMedia instruction set, or postponed for a subsequent stage of MPEG decoding process. We mention that when the IDCT rounding and saturation is carried out immediately after the 2-D IDCT completed, the square of the intrinsic Loeffler gain is also compensated out by right-shifting by three positions (i.e., integer division by 8).

As expected, the best result is obtained for a software pipeline loop: 1/32 IDCT/cycle. However, a linear loop organization with two FPGA-based IDCTs and $2\times$ unrolling is not a bad choice either, since it achieves a throughput only 17% lower: 1/37.3 IDCT/cycle. Since generating software pipeline loops is not supported by the current TriMedia toolchain, the advantage of the later approach is an easier programming task. If IDCT rounding and saturation can be postponed for a different stage of MPEG decoding process, the best solution for a linear loop corresponds to a computing scenario with four FPGA-based and one software-based IDCTs in the loop. The throughput in this case is 1/35.6 IDCT/cycle. For the same linear loop organization and a computing scenario in which the vertical 1-D IDCTs are computed for all the matrices of the testbench in a first loop, and only then all the horizontal 1-D IDCTs are carried out in a separate loop, the double overhead associated to the prologues and epilogues of the two loops decreases the throughput to 1/42.8 IDCT/cycle (about 6% lower). Unrolling the loop three or more times generates register spilling; thus the performance degrades significantly.

It is worth noting that IDCT rounding and saturation carried out in software requires about 1.5 cycles/IDCT, and only $0.5 \div 1.0$ cycles/IDCT when carried out in FPGA. However, two `RFU-OP-IDs` are needed to embed IDCT rounding and saturation in FPGA: one `ID` for horizontal 1-D IDCT, and one `ID` for vertical 1-D IDCT. At this moment we would like to emphasize that due to the MOLEN concept which provides means to specify multiple RFU-based operations for the same `RFU-OP`, the need for two or more `IDs` never becomes a limitation.

In Table 6 and Fig. 21 we compare the performances of several 2-D IDCT implementations: on standard TriMedia [1], on FPGA-augmented TriMedia, on FPGA alone [19], and on several 2-D convolution-oriented coarse-grain programmable architectures: REMARC [20], MorphoSys [21], M.F.A.S.T. [23], and ManArray [22]. Since the IDCT is applied on large batches of 8×8 blocks, the throughput is more important than latency. For this reason, our performance analysis is focused on the 2-D IDCT throughput. A special remark regarding the *distributed arithmetic*-based implementation on FPGA alone has to be made. Since the multiplications are computed by looking-up into on-chip small memories (the so called Block SelectRAM cells) [19], the pipeline cannot be made deeper, and 55.6 MHz is the upper bound of the frequency that can be achieved

Table 5. Performance figures for 8 × 8 IDCT on (FPGA-augmented) TriMedia.

Loop organization	Computing scenario	Unrolling degree	IDCT rounding and saturation	Effectiveness (issues/cycle)	Performance (cycles/ 8 × 8 matrix)	Comments
Linear	Two FPGA-IDCTs	None	None	1.94	41.5	
Linear	Two FPGA-IDCTs	None	In FPGA	1.87	43.0	Requires two RFU-OP-IDs
Linear	Two FPGA-IDCTs	None	In SW	1.87	44.4	
Linear	Two FPGA-IDCTs	2×	None	2.18	36.8	
Linear	Two FPGA-IDCTs	2×	In FPGA	2.15	37.3	Requires two RFU-OP-IDs
Linear	Two FPGA-IDCTs	2×	In SW	3.76	38.5	
Linear	Two FPGA-IDCTs	3×	None	2.30	61.6	Spilling encountered
Linear	Two FPGA-IDCTs	3×	In FPGA	2.11	63.1	Spilling encountered Requires two RFU-OP-IDs
Linear	Two FPGA-IDCTs	3×	In SW	3.22	64.0	Spilling encountered
Linear	Four FPGA-IDCTs + one SW-IDCT	None	None	3.10	35.6	
Linear	Four FPGA-IDCTs + one SW-IDCT	None	In FPGA/SW	3.24	38.2	Requires two RFU-OP-IDs
Linear	16 Vertical 1-D IDCTs + Transposition	None	n/a	2.30	28.0	
	16 Horizontal 1-D IDCTs	None	None	1.90	25.5	
	Total for 2-D IDCT	–	–	–	53.5	
Linear	16 Vertical 1-D IDCTs + Transposition	None	n/a	2.30	28.0	Requires two RFU-OP-IDs
	16 Horizontal 1-D IDCTs	None	In FPGA	1.83	26.5	
	Total for 2-D IDCT	–	–	–	54.5	
Linear	16 Vertical 1-D IDCTs + Transposition	2×	n/a	2.92	22.0	
	16 Horizontal 1-D IDCTs	2×	None	2.33	20.8	
	Total for 2-D IDCT	–	–	–	42.8	
Linear	16 Vertical 1-D IDCTs + Transposition	2×	n/a	2.92	22.0	Requires two RFU-OP-IDs
	16 Horizontal 1-D IDCTs	2×	In FPGA	2.27	21.3	
	Total for 2-D IDCT	–	–	–	43.3	
Linear	16 Vertical 1-D IDCTs + Transposition	3×	n/a	2.79	37.2	Spilling encountered
	16 Horizontal 1-D IDCTs	3×	None	2.03	35.5	Requires two RFU-OP-IDs
	Total for 2-D IDCT	–	–	–	72.7	

(*Continued on next page.*)

Table 5. (*Continued*).

Loop organization	Computing scenario	Unrolling degree	IDCT rounding and saturation	Effectiveness (issues/cycle)	Performance (cycles/ 8 × 8 matrix)	Comments
Linear	16 Vertical 1-D IDCTs + Transposition	3×	n/a	2.79	37.2	Spilling encountered
	16 Horizontal 1-D IDCTs	3×	In FPGA	1.98	36.8	Requires two RFU-OP-IDs
	Total for 2-D IDCT	–	–	–	74.0	
Software pipelined	Two FPGA-IDCTs	n/a	None	3.81	32.1	
Software pipelined	Two FPGA-IDCTs	n/a	In FPGA	3.69	33.1	Requires two RFU-OP-IDs
Software pipelined	All Vertical 1-D IDCTs + Transposition	n/a	n/a	3.20	20.1	
	All Horizontal 1-D IDCTs	n/a	None	4.10	16.1	
	Total for 2-D IDCT	–	–	–	36.2	
Software pipelined	All Vertical 1-D IDCTs + Transposition	n/a	n/a	3.20	20.1	Requires two RFU-OP-IDs
	All Horizontal 1-D IDCTs	n/a	In FPGA	4.10	16.2	
	Total for 2-D IDCT	–	–	–	36.3	

Note: FPGA-IDCT stands for an IDCT which benefits from reconfigurable hardware support.
SW-IDCT stands for an IDCT carried out in software.

Table 6. Performance figures for IDCT on several high-performance architectures.

		Throughput		Latency		
Implementation	FPGA family	IDCT/cycle	IDCT/sec	Cycles	ns	FPGA utilization
Standard TriMedia (200 MHz) [1]	n/a	1/56	3.57 M	56	280	n/a
FPGA-augmented TriMedia	EP1K100 (Altera)	1/32	6.25 M	42	210	45%
FPGA alone (55.6 MHz) [19]	XCV600 (Xilinx)	Non relevant	4.27 M	Non relevant	467.9	88%
REMARC [20]	Coarse grain	1/54	No	54	No info.	100%
MorphoSys (100 MHz) [21]	Coarse grain	1/37	2.70 M	37	370	100%
M.F.A.S.T. (50 MHz) [23]	n/a	1/22	2.27 M	22	440	n/a
ManArray [22]	n/a	1/34	No	34	No info.	n/a

on the Virtex XCV600 device for such implementation. Thus, the number of cycles which corresponds to a throughput of 4.27 millions IDCT/sec at a clock frequency of 55.6 MHz is not relevant, and the comparison with the processor-based implementations has to be made in terms of absolute throughput expressed in IDCT/sec. With 6.25 millions IDCT/sec, the FPGA-augmented TriMedia provides an improvement of 46% in terms of throughput over FPGA alone.

The 2-D IDCT implementation on standard TriMedia exhibits the lowest throughput (1/56 IDCT/cycle), while the highest throughput (1/22 IDCT/cycle) is achieved for the implementation on M.F.A.S.T. The second highest throughput (1/32 IDCT/cycle) is achieved for the implementation on augmented TriMedia, which is an improvement of 75% over standard TriMedia (40% in terms of computing time). We would like to comment that the difference in performance between M.F.A.S.T. and FPGA-augmented TriMedia will diminish if additional computation is considered, e.g., the post-IDCT rounding and saturating required by MPEG standard [5]. While the throughput will decrease on M.F.A.S.T., it will remain about the same on FPGA-augmented TriMedia, since the 1-D IDCT pipeline can be easily enlarged by an additional stage for computing post-IDCT rounding

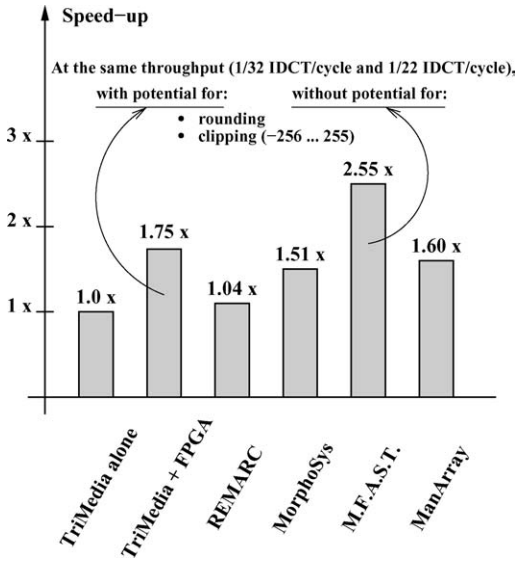

Figure 21. The speed-up of various IDCT implementations on several high-performance architectures relative to standard TriMedia.

and saturating (see Table 5). Alternatively, post-IDCT rounding and saturating can be implemented within the standard instruction set of TriMedia, since there are still plenty of empty slots, as already mentioned.

Finally, we would like to mention that the second highest throughput is achieved with a fine-grain field-programmable custom computing machine, that is, FPGA-augmented TriMedia, which exhibits flexibility over a 2-D convolution-oriented architectures like M.F.A.S.T. [23] or ManArray [22] for implementing heterogenous tasks, e.g., variable-length decoding [5].

5. Conclusions

We have proposed an architectural extension for TriMedia which encompasses an FPGA–based Reconfigurable Functional Unit, a hardwired Configuration Unit managing the reconfiguration of the FPGA, and the associated instructions. On an FPGA-augmented TriMedia/CPU64, we obtained a performance improvement of 75% in terms of throughput over standard TriMedia–CPU64 for an 8×8 IDCT, at the expense of two new instructions (`SET` and `EXECUTE`), and of a medium-size FPGA (on the order of 5,000 4-input LUTs). Since we made the conservative assumption that the FPGA clock frequency is at most half the TriMedia clock frequency, the announced figure represents the lower bound of the performance improvement for the IDCT which can be achieved on FPGA-augmented TriMedia. Given the fact that the experimental TriMedia is a 5 issue-slot VLIW processor with 64-bit datapaths and a very rich multimedia instruction set, such an improvement within its target media processing domain indicates that indicates that augmenting TriMedia–CPU64 with an FPGA shows clear benefit for doing 2-D IDCT.

Acknowledgments

The authors would like to thank Dr. Evert-Jan Pol with Philips Semiconductor for numerous helpful comments. This project was supported by the doctoral fellowship RWC-061-PS-99047-ps from Philips Research Laboratories in Eindhoven, The Netherlands.

Notes

1. i.e., the operation slot-width and the number of input and output registers.
2. i.e., the issuing slot(s) that the computing facility is sensitive to.

References

1. J.T.J. van Eijndhoven, F.W. Sijstermans, K.A. Vissers, E.-J.D. Pol, M.J.A. Tromp, P. Struik, R.H.J. Bloks, P. van der Wolf, A.D. Pimentel, and H.P. Vranken, "TriMedia CPU64 Architecture," in *Proceedings of International Conference on Computer Design (ICCD '99)*, Austin, Texas, 1999, pp. 586–592.
2. J.T. van Eijndhoven and F. Sijstermans, "Data Processing Device and method of Computing the Cosine Transform of a Matrix," U.S. Patent No. 6,397,235, 2002.
3. A.K. Riemens, K.A. Vissers, R.J. Schutten, F.W. Sijstermans, G.J. Hekstra, and G.D.L. Hei, "TriMedia CPU64 Application Domain and Benchmark Suite," in *Proceedings of International Conference on Computer Design (ICCD '99)*, Austin, Texas, 1999, pp. 580–585.
4. K.R. Rao and P. Yip, *Discrete Cosine Transform. Algorithms, Advantages, Applications*, San Diego, California: Academic Press, 1990.
5. J.L. Mitchell, W.B. Pennebaker, C.E. Fogg, and D.J. LeGall, *MPEG Video Compression Standard*, New York, New York: Chapman & Hall, 1996.
6. C. Loeffler, A. Ligtenberg, and G.S. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '89)*, 1989, pp. 988–991.
7. "IEEE Standard Specifications for the Implementations of 8×8 Inverse Discrete Cosine Transform," *IEEE Std* 1991, pp. 1180–1990.
8. S. Brown and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Transactions on Design and Test of Computers* vol. 13, no. 2, 1996, pp. 42–57.

9. Altera Corporation, *ACEX 1K Programmable Logic Family*, Datasheet, San Jose, California, 2000.
10. J.T. van Eijndvhoven, G.A. Slavenburg, and S. Rathnam, "VLIW Processor has Different Functional Units Operating on Commands of Different Widths," U.S. Patent No. 6,076,154, 2000.
11. S. Vassiliadis, S. Wong, and S. Cotofana, "The MOLEN $\rho\mu$-coded Processor," in *11th International Conference on Field-Programmable Logic and Applications (FPL 2001)*, vol. 2147 of *Lecture Notes in Computer Science (LNCS)*, Belfast, Northern Ireland, United Kingdom, Springer-Verlag, 2001, pp. 275–285.
12. M. Sima, S. Vassiliadis, S.D. Cotofana, J.T. van Eijndhoven, and K.A. Vissers, "Field-Programmable Custom Computing Machines. A Taxonomy," in *12th International Conference on Field-Programmable Logic and Applications (FPL 2002)*, vol. 2438 of *Lecture Notes in Computer Science (LNCS)*, Montpellier, France, Springer-Verlag, 2002, pp. 79–88.
13. E.-J.D. Pol, B.J.M. Aarts, J.T.J. van Eijndhoven, P. Struik, F.W. Sijstermans, M.J.A. Tromp, J.W. van de Waerdt, and P. van der Wolf, "TriMedia CPU64 Application Development Environment," in *Proceedings of International Conference on Computer Design (ICCD '99)*, Austin, Texas, 1999, pp. 593–598.
14. A. DeHon, "Reconfigurable Architectures for General-Purpose Computing," A. I. 1586, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1996.
15. J. van Eijndhoven, "16-Bit Compliant Software IDCT on TriMedia/CPU64," Internal Report NL-TN 171, Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands, 1997.
16. M. Sima, S. Cotofana, J.T. van Eijndhoven, S. Vassiliadis, and K. Vissers, "8 $\times$ 8 IDCT Implementation on an FPGA-Augmented TriMedia," in *9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, Rohnert Park, California, 2001.
17. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, New York, New York: Oxford University Press, 2000.
18. J. Hoogerbrugge and L. Augusteijn, "Instruction Scheduling for TriMedia," *Journal of Instruction-Level Parallelism*, vol. 1, no. 1, 1999.
19. K. Chaudhary, H. Verma, and S. Nag, "An Inverse Discrete Cosine Transform (IDCT) Implementation in Virtex for MPEG Video Application," Application Note 208, Xilinx Corporation, San Jose, California, 1996.
20. T. Miyamori and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor," *IEEE Transactions on Information and Systems*, vol. E82-D, no. 2, 1999, pp. 389–397.
21. H. Singh, M.-H. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh, and E.M.C. Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Application," *IEEE Transactions on Computers*, vol. 49, no. 5, 2000, pp. 465–481.
22. G.G. Pechanek and S. Vassiliadis, "The ManArray Embedded Processor Architecture," in *Proceedings of the 26th Euromicro Conference, "Informatics: Inventing the Future"*, Maastricht, The Netherlands, 2000, pp. 348–355.
23. G.G. Pechanek, C.W. Kurak, C.J. Glossner, C.H.L. Moller, and S.J. Walsh, "M.F.A.S.T.: A Highly Parallel Single Chip DSP with a 2D IDCT Example," in *Proceeding of the International Conference on Signal Processing Applications and Technology (ICSPAT '95)*, Boston, Massachusetts, 1995, pp. 69–72.

Mihai Sima was born in Bucharest, Romania. He received the MS degree in Electrical Engineering from 'Politehnica' University of Bucharest, and the Ph.D. degree in Electrical Engineering from Delft University of Technology, The Netherlands. He had been with the 'Microelectronics' Company in Bucharest for 3 years, where he was involved in instrumentation electronics for integrated circuit testing. Subsequently, he joined the Telecommunications Department of 'Politehnica' University of Bucharest, where he had been involved in digital signal processing and speech recognition for 6 years. More recently, he had been with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, where he worked on reconfigurable architectures for media-processing domain. He is currently an assistant professor with the Department of Electrical and Computer Engineering, University of Victoria, B.C., Canada. His research interests include computer architecture, reconfigurable computing, embedded systems, digital signal processing, and speech recognition.
msima@ece.uvic.ca

Sorin D. Coţofană was born in Mizil, Romania. He received the MS degree in Computer Science from the 'Politehnica' University of Bucharest, Romania, and the Ph.D. degree in Electrical Engineering from Delft University of Technology, The Netherlands. He had worked with the Research & Development Institute for Electronic Components (ICCE) in Bucharest for a decade, being involved in structured design of digital systems, design rule checking of IC's layout, logic and mixed-mode simulation of electronic circuits, testability analysis, and image processing. He is currently an associate professor with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, The Netherlands. His research interests include computer arithmetic, parallel architectures, embedded systems, reconfigurable computing, nano-electronics, neural networks, computational geometry, and computer aided design.
s.d.cotofana@ewi.tudelft.nl

Jos T.J. van Eijndhoven was born in Roosendaal, The Netherlands. He studied Electrical Engineering at the Eindhoven University of Technology, The Netherlands, obtaining the M.Sc. and Ph.D. degrees in 1981 and 1984, respectively, for a work on piecewise linear circuit simulation. Then, he became a senior research member in the design automation group of the Eindhoven University of Technology. In 1986 he spent a sabbatical period at the IBM Thomas J. Watson Research Laboratory, Yorktown Heights, New York, for research on high level synthesis. In 1998 he joined Philips Research Laboratories in Eindhoven, The Netherlands, to work on the architectural design of programmable multimedia hardware and the associated mapping of media processing applications.
jos.van.eijndhoven@philips.com

Stamatis Vassiliadis was born in Manolates, Samos, Greece. He is a professor with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, The Netherlands. He has also served in the faculties of Cornell University, Ithaca, NY, and the State University of New York (S.U.N.Y.), Binghamton, NY. He had worked for a decade with IBM in the Advanced Workstations and Systems laboratory in Austin TX, the Mid-Hudson Valley Laboratory in Poughkeepsie, NY, and the Glendale Laboratory in Endicott, NY. In IBM he was involved in a number of projects regarding computer design, organizations, and architectures and the leadership to advanced research projects. A number of his design and implementation proposals have been implemented in commercially-available systems and processors including the IBM 9370 model 60 computer system, the IBM POWER II, the IBM AS/400 Models 400, 500, and 510, Server Models 40S and 50S, the IBM AS/400 Advanced 36, and the IBM S/390 G4 and G5 computer systems. For his work, he received numerous awards including 23 levels of Publication Achievement Awards, 15 levels of Invention Achievement Awards and an Outstanding Innovation Award for Engineering/Scientific Hardware Design in 1989. In 1990 he has been awarded the highest number of USA patents in IBM, six of his 70 USA patents being rated with the highest patent ranking in IBM.
s.vassiliadis@ewi.tudelft.nl

Kees A. Vissers graduated the Delft University of Technology, receiving his M.Sc. in 1980. He started directly with Philips Research Laboratories in Eindhoven where he was involved in high-level simulation and high-level synthesis. He had been heading the research on hardware/software co-design and system level design for many years, and had a significant contribution to the TriMedia VLIW processor. From 1987 till 1988 he was a visiting researcher at Carnegie Mellon University, Pittsburgh, Pennsylvania, with the group of Don Thomas. He is currently a Research Fellow with University of California at Berkeley, Department of Electrical Engineering and Computer Sciences. His research interests include video processing, embedded media processing systems, and reconfigurable computing.
vissers@eecs.berkeley.edu