Proceedings of PROGRESS 2001, Veldhoven, The Netherlands, October 18, 2001, pp. 211-218 (!)

An 8-Point IDCT Computing Resource Implemented on a TriMedia/CPU64 Reconfigurable Functional Unit

Mihai Sima[†‡] Sorin Cotofana[†] Jos T.J. van Eijndhoven[‡] Stamatis Vassiliadis[†]
[†]*Delft University of Technology, – Dept. of Electrical Engineering, Delft, The Netherlands*
[‡]*Philips Research, – Dept. of Information and Software Technology, Eindhoven, The Netherlands*
Phone: +31-(0)40-274-2593 E-mail: M.Sima@et.tudelft.nl

Abstract— **This paper presents the implementation of an 8-point Inverse Discrete Cosine Transform (IDCT) computing resource on a TriMedia/CPU64 FPGA-based Reconfigurable Functional Unit (RFU). TriMedia/CPU64 is a 64-bit 5 issue-slot VLIW processor launching a long instruction every clock cycle. The RFU consists mainly of an FPGA core, and is embedded into the TriMedia as any other hardwired functional unit, i.e., it receives instructions from the instruction decoder, reads its input arguments from and writes the computed values back to the register file. To reduce the computational complexity of IDCT, we used a modified version of the Loeffler algorithm which requires 14 multiplications. Since each multiplicand is a 16-bit signed number represented in 2's complement notation, while each multiplier is a positive constant of 15 bits or less, we employed a "multiplication-by-constant" scheme which was optimized against the multiplier. To increase the throughput of the IDCT computing resource, we propose a pipeline implementation. When mapped on an ACEX EP1K100 FPGA-based RFU, our 8-point IDCT computing resource exhibits a latency of 16 TriMedia cycles, a recovery of 2 cycles, and occupies 42% of the logic cells of the device.**

Keywords— **Reconfigurable computing, inverse discrete cosine transform, VLIW processors, field-programmable gate array.**

I. 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 [1], [2], [3], [4], [5], [6], [7]. 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. This paper presents the implementation of an 8-point Inverse Discrete Cosine Transform (IDCT) computing resource on a TriMedia/CPU64 FPGA-based Reconfigurable Functional Unit (RFU). TriMedia/CPU64 is a 64-bit 5 issue-slot VLIW processor launching a long instruction every clock cycle. The RFU consists mainly of an FPGA core, and is embedded into the TriMedia as any other hardwired functional unit, i.e., it receives instructions from the instruction decoder, reads its input arguments from and writes the computed values back to the register file. With such RFU, the user can define and use any computing facility subject to the FPGA size and TriMedia organization.

IDCT is a highly computational intensive part of MPEG decoding, and is used in the JPEG decompression of data as well. To reduce the computational complexity of IDCT, we used a modified version of the Loeffler algorithm [11] which requires 14 multiplications. Since each multiplicand is a 16-bit signed number represented in 2's complement notation, while each multiplier is a positive constant of 15 bits or less, we employed a "multiplication-by-constant" scheme which was optimized against the multiplier. In order to increase the throughput of the IDCT computing resource, we propose a pipeline implementation. When mapped on an ACEX EP1K100 FPGA-based RFU, our 8-point IDCT computing resource exhibits a latency of 16 TriMedia cycles, a recovery of 2 cycles, and occupies 42% of the logic cells of the device.

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 II. Section III briefly describes the architectural extension of TriMedia/CPU64. Implementation issues of the 1-D IDCT computing resource on FPGA, as well as experimental results are presented in Section IV. Section V completes the paper with some conclusions and closing remarks.

II. 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 Inverse Discrete Cosine Transform has found wide applications in image processing, data compression, filtering, and other fields. The transformation for an N point 1-D IDCT is defined by [9]:

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

where X_u are the inputs, x_i are the outputs, and $K_u = \sqrt{1/2}$ for $u = 0$, otherwise is 1. One of the most efficient algorithms for computing an 8-point IDCT has been proposed by Loeffler [10]. 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 [11]. Subsequently, we will use this modified algorithm (see Figure 1).

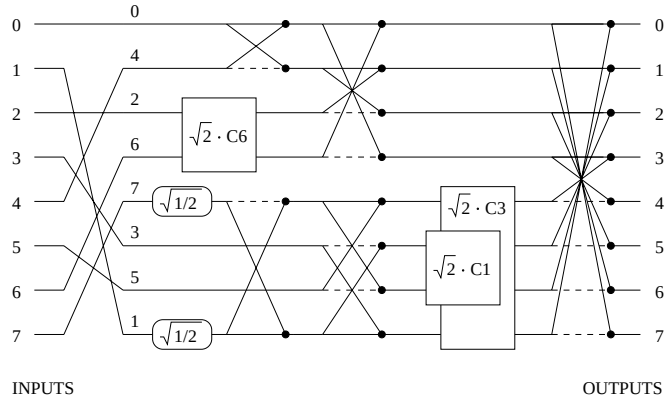
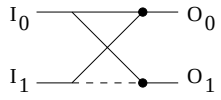


Fig. 1. The modified 'Loeffler' algorithm – [11].

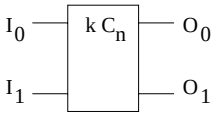
In the figure, the round block signifies a multiplication by $C'_0 = \sqrt{1/2}$. The butterfly block and the associated equations are presented in Figure 2.



$$\begin{aligned} O_0 &= I_0 + I_1 \\ O_1 &= I_0 - I_1 \end{aligned}$$

Fig. 2. The butterfly – [10].

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 3.



$$O_0 = I_0 k \cos \frac{n\pi}{16} - I_1 k \sin \frac{n\pi}{16} = C'_n I_0 - S'_n I_1$$
$$O_1 = I_0 k \sin \frac{n\pi}{16} + I_1 k \cos \frac{n\pi}{16} = S'_n I_0 + C'_n I_1$$

Fig. 3. The rotator – [10].

Although an implementation of such a rotator with three multiplications and three additions is possible (Fig. 4 – a, 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) [12] 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 configuration memory. In the sequel, we will assume that the architecture of the raw hardware is identical with that of an ACEX 1K device from Altera [13]. 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 for EP1K10 device to 4992 logic cells 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 [13].

The next section describes the architectural extension for the TriMedia-CPU64.

III. TriMedia architectural extension

TriMedia-CPU64 is a 64-bit 5 issue-slot VLIW core, launching a long instruction every clock cycle [8]. 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 every clock cycle. In addition, each operation can be guarded with an optional (4^{th}) register for conditional execution without branch penalty. The architecture supports subword parallelism and is optimized with respect to media-processing. With the exception of

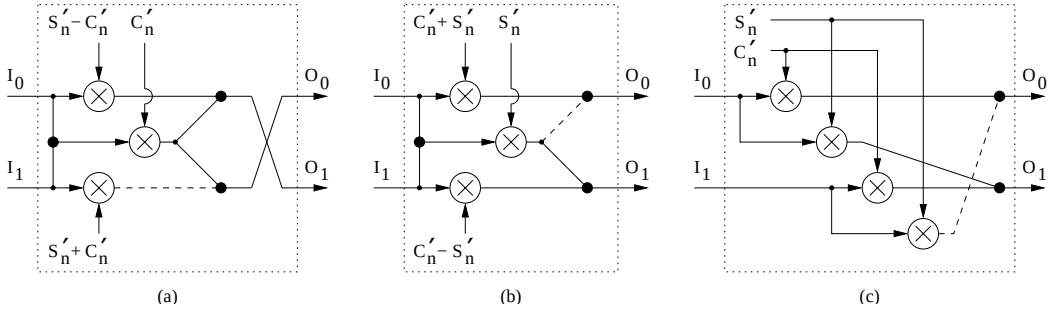


Fig. 4. Three possible implementations of the rotator

floating point divide and square root unit, all functional units have a recovery of 1, while their latency ranges from 1 to 4 The TriMedia-CPU64 VLIW core also supports double-slot operations, or super-operations. Such a super-operation occupies two adjacent slots in the VLIW instruction, and maps to a double-width functional unit. This way, operations with more than 2 arguments and one result are possible. The current organization of the TriMedia/CPU64 core is presented in Figure 5.

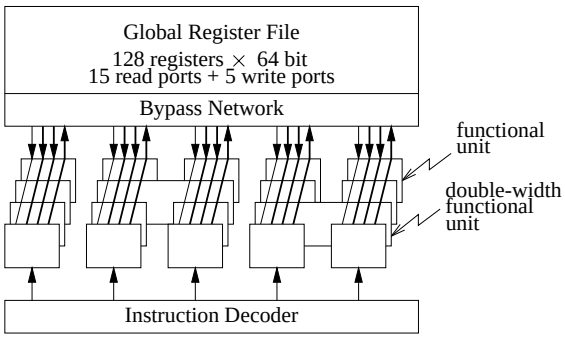


Fig. 5. TriMedia/CPU64 organization – [8].

In the sequel, we will assume that the TriMedia/CPU64 processor is augmented with a Reconfigurable Functional Unit (RFU) which consists mainly of a reconfigurable array core. A hardwired Configuration Unit which manages the reconfiguration of the raw hardware is associated to the RFU, as depicted in Figure 6. The reconfigurable functional unit is embedded into TriMedia as any other hardwired functional unit, 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, and also of the associated compiler and scheduler 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 [14]. Generally speaking, the reconfiguration of the RFU is performed under the command of a `SET` instruction, while `EXECUTE` instructions launch the operations to be performed by the computing resources configured on the raw hardware [15]. In this way, the execution of an RFU-mapped operation requires two basic stages: `SET` and `EXECUTE`.

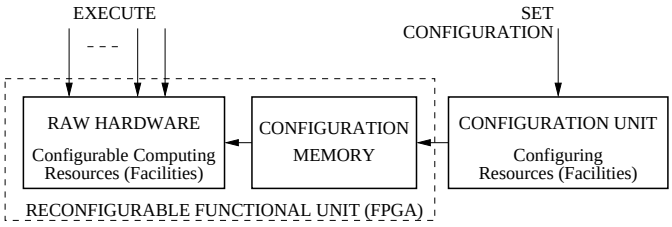


Fig. 6. The architectural extension for TriMedia/CPU64 VLIW core – [14].

With such architectural extension, the user is given the freedom to define and use any computing facility subject to the FPGA size and TriMedia organization. In the sequel, a single RFU-related operation which computes an 8-point IDCT will be defined. We will present the implementation details of the 8-point IDCT computing facility on the FPGA, the achieved performance, and the associated costs in terms of FPGA area.

IV. Experimental results

As mentioned, an 8-point IDCT 2-slot computing resource will be mapped on the RFU. By launching an 8-point IDCT super-operation having two 64-bit inputs and two 64-bit outputs, an 8-point 1-D IDCT is computed on eight 16-bit values. The RFU is configured at application launch-time, i.e., a `SET` instruction is scheduled on the top of the program code of the application. An ACEX 1K FPGA from Altera will be used as an experimental platform for the reconfigurable core.

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 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 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 II, and to Figures 1, 3, and 4, since the 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.

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 useless, and a customized implementation of the scheme will lead to an 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 a certain FPGA family may not be appropriate for a different family.

In connection with the partial product matrix, measured performances of several reduction modules for ACEX 1K are presented in Table I. All the values in the table correspond to synchronous designs, i.e., both inputs and outputs are registered. The estimations have been obtained by compiling VHDL source codes with Leonardo Spectrum™from Exemplar, followed by a place and route procedure performed by MAX+PLUS II™from Altera. Since the maximum operating frequency for the ACEX 1K is 180 MHz for the time being, we will proceed to a conservative assumption and consider the figures typed in italics as being too optimistic, although they are generated by software tools. 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)*.

TABLE I
PERFORMANCES OF SEVERAL REDUCTION MODULES FOR ACEX 1K 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*

In order to implement an IDCT at 100 MHz, reduction modules which can run at 100 MHz or more should be considered. These modules are summarized below:

- Horizontal reductions of two, 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).

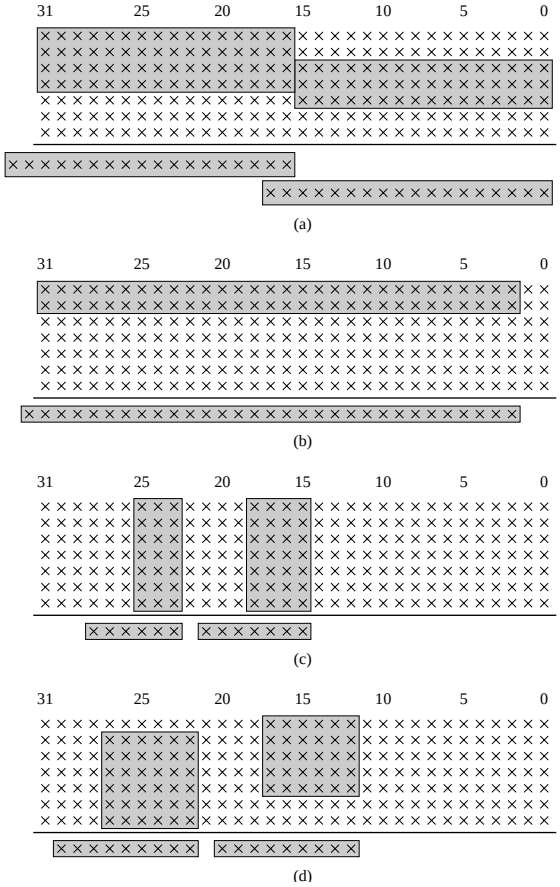


Fig. 7. 100 MHz reduction modules on ACEX 1K.

As a general rule, a horizontal reduction module consumes a lower area than a vertical reduction module of the same size. This situation occurs because a horizontal reduction module can intensively use the carry chain, while a vertical reduction module cannot. A second observation is that Dadda population counters [16] of 3 or 4 bits can be implemented in only one logic level, i.e., with a delay of 0.6 ns [13] with two, respectively three LUTs. Also, Dadda counters of 5 or 6 bits can be implemented in two cascaded logic levels which exhibit a total delay of 1.2 ns, with seven LUTs. Although Dadda counters could theoretically be used as a reduction technique working at the same frequency with TriMedia, i.e., minimum 200 MHz, such an approach is limited by the maximum operating frequency of the ACEX 1K FPGAs: 180 MHz.

Reduction modules which can run at 100 MHz have been determined. Regarding the implementation we will 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 [17]. Also, only the most significant 16 bits of the product have to be stored.

The partial product matrix and the selected reduction modules and steps for multiplication by the constant $C'_0 = 5\mathrm{a}82\,\mathrm{h}$ are presented in Figure 8 ('S' represents the sign-bit). In the first step, the partial product matrix is built. Then, reductions on the modules specified by the shaded areas are carried out. The 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.

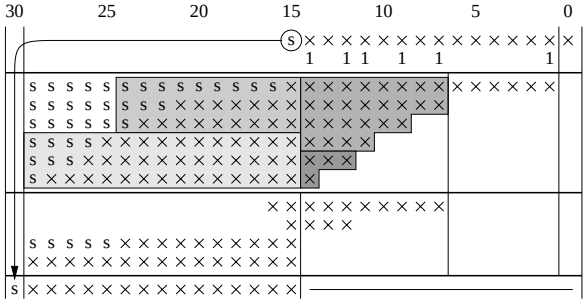


Fig. 8. The partial product matrix and the selected reduction steps for multiplication by the constant C'_0

The partial product matrix and the selected reduction modules and steps for multiplication by the constant $C'_1 = 58\mathrm{c}5\,\mathrm{h}$ are presented in Figure 9. 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.

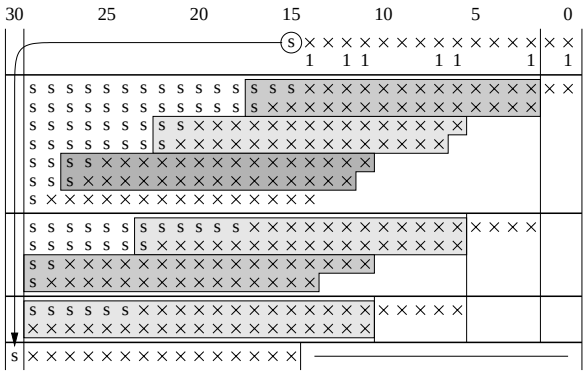


Fig. 9. The partial product matrix and the selected reduction steps for multiplication by the constant C'_1

The partial product matrix and the selected reduction modules and steps for multiplication by the constants $S'_1 = 11\mathrm{a}8\,\mathrm{h}$, $C'_3 = 4\mathrm{b}42\,\mathrm{h}$, $S'_3 = 3249\,\mathrm{h}$, $C'_6 = 22\mathrm{a}3\,\mathrm{h}$, and $S'_6 = 539\mathrm{f}\,\mathrm{h}$ are presented in Figures 10, 11, 12, 13, and 14, respectively. Concerning 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, consequently, the number of rows in the corresponding partial product matrix, the Booth's recoding [16] will pe 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 $0081\,\mathrm{h}$ will be subtracted rather than added when being reduced.

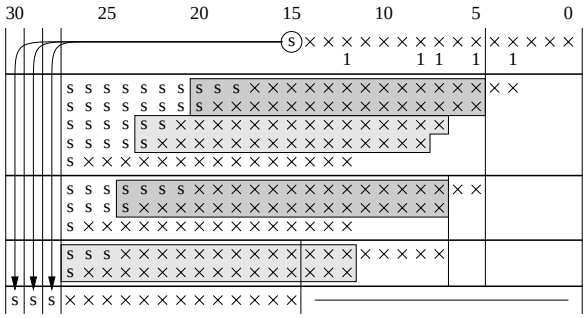


Fig. 10. The partial product matrix and the selected reduction steps for multiplication by the constant S'_1

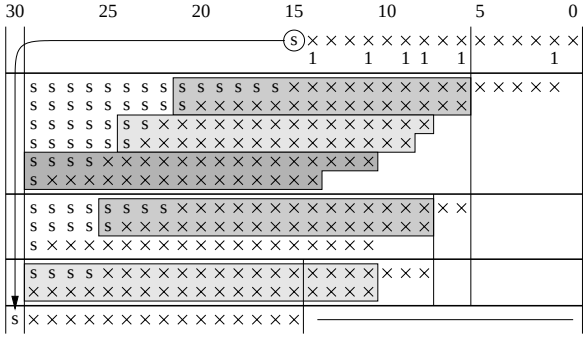


Fig. 11. The partial product matrix and the selected reduction steps for multiplication by the constant C'_3

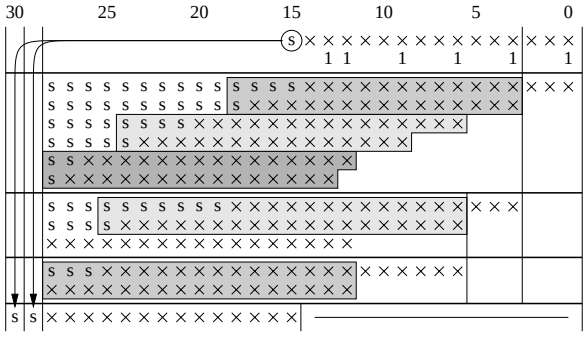


Fig. 12. The partial product matrix and the selected reduction steps for multiplication by the constant S'_3

We would like to note that the critical path of the 1-D IDCT is located on the odd part of the modified 'Loeffler' algorithm. Once the multiplication by constant C'_1 is performed in three stages, there is no gain in performance to implement the other three multiplications, i.e., 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 (however, at the expense of a slightly larger area). The same considerations apply for multiplications by the constants C'_6 and S'_6, as both of them are not located on the critical path.

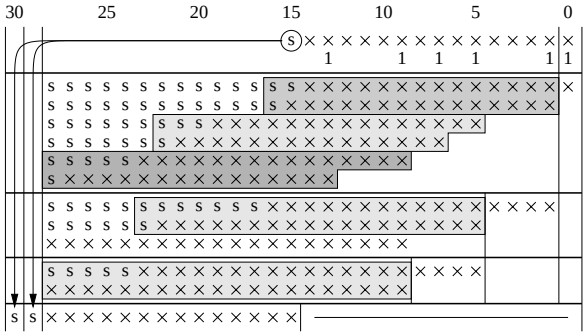


Fig. 13. The partial product matrix and the selected reduction steps for multiplication by the constant C'_6

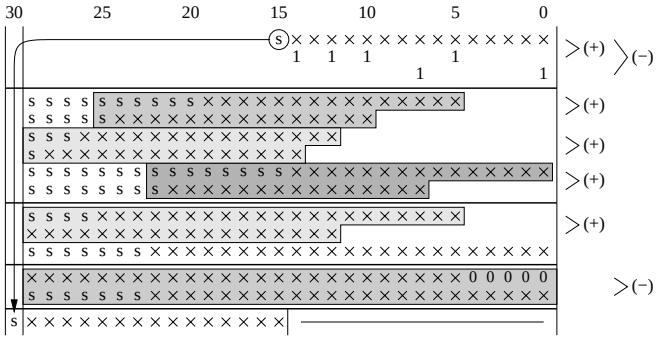


Fig. 14. The partial product matrix and the selected reduction steps for multiplication by the constant S'_6

The sketch of the 1-D IDCT pipeline is depicted in Figure 15 (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 1-D IDCT's output flip-flops to the register file.

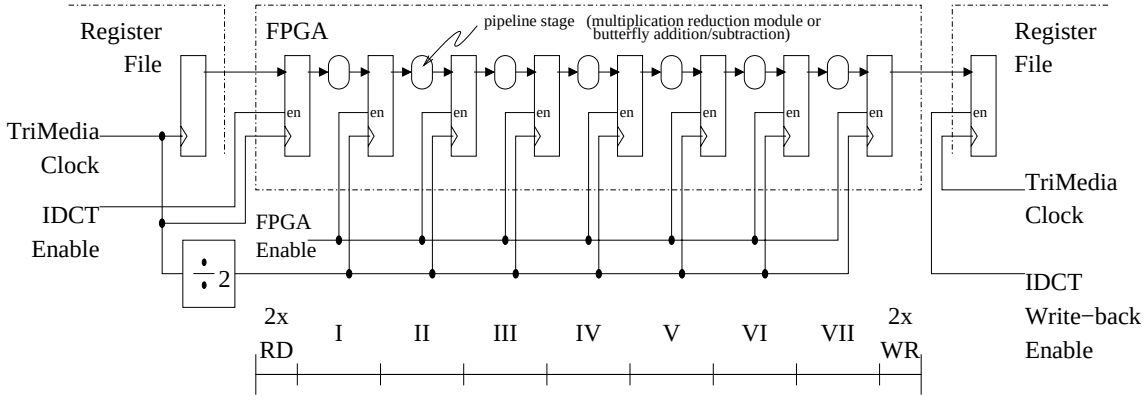


Fig. 15. The 1-D IDCT pipeline

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.

V. Conclusions and future work

We described an architectural extension for TriMedia/CPU64 which encompasses a reconfigurable functional unit and the associated instructions. We proved that a resource which can compute the 8-point IDCT in 16 TriMedia cycles can be configured on the reconfigurable functional unit. 42% of the logic cells of an EP1K100 device has been used to implement the 8-point IDCT. In future work, we intend to implement an appropriate rounding scheme in order to fulfill the IEEE numerical accuracy requirements for IDCT in MPEG applications.

References

[1] Rahul Razdan and Michael D. Smith, "A High Performance Microarchitecture with Hardware-Programmable Functional Units," in *27th Annual International Symposium on Microarchitecture (MICRO-27)*, San Jose, California, November 1994, pp. 172–180.

[2] Ralph D. Wittig and Paul Chow, "OneChip: An FPGA Processor With Reconfigurable Logic," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, Napa Valley, California, April 1996, pp. 126–135.

[3] John R. Hauser and John Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '97)*, Napa Valley, California, April 1997, pp. 12–21.

[4] Takashi Miyamori and Kunle Olukotun, "A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '98)*, Napa Valley, California, April 1998, pp. 2–11.

[5] Sergej Sawitzki, Achim Gratz, and Rainer G. Spallek, "Increasing Microprocessor Performance with Tightly-Coupled Reconfigurable Logic Arrays," in *Field-Programmable Logic and Applications: From FPGAs to Computing Paradigm. 8th International Workshop (FPL '98)*, Tallin, Estonia, September 1998, vol. 1482 of *Lecture Notes in Computer Science*, pp. 411–415.

[6] Jeffrey A. Jacob and Paul Chow, "Memory Interfacing and Instruction Specification for Reconfigurable Processors," in *ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays (FPGA '99)*, Monterey, California, February 1999, pp. 145–154.

[7] Bernardo Kastrup, A. Bink, and J. Hoogerbrugge, "ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '99)*, Napa Valley, California, April 1999, pp. 92–100.

[8] Jos 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. E. Vranken, "TriMedia CPU64 Architecture," in *Proceedings of International Conference on Computer Design (ICCD '99)*, Austin, Texas, October 1999, pp. 586–592.

[9] K. R. Rao and P. Yip, *Discrete Cosine Transform. Algorithms, Advantages, Applications*, Academic Press, San Diego, California, 1990.

[10] Christoph Loeffler, Adriaan Ligtenberg, and George 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.

[11] Jos van Eijndhoven and Frans Sijstermans, "Data Processing Device and method of Computing the Cosine Transform of a Matrix," PCT Patent No. WO 9948025, September 1999.

[12] Stephen Brown and Jonathan Rose, "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Transactions on Design and Test of Computers*, vol. 13, no. 2, pp. 42–57, 1996.

[13] Altera Corporation, "ACEX 1K Programmable Logic Family," Datasheet, San Jose, California, April 2000.

[14] Mihai Sima, Sorin Cotofana, Jos T.J. van Eijndhoven, Stamatis Vassiliadis, and Kees Vissers, "8×8 IDCT Implementation on an FPGA-augmented TriMedia," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 2001)*, Rohnert Park, California, April 2001.

[15] Mihai Sima, Stamatis Vassiliadis, Sorin Cotofana, Jos T.J. van Eijndhoven, and Kees Vissers, "A Taxonomy of Custom Computing Machines," in *Proceedings of the First PROGRESS Workshop on Embedded Systems*, Jean Pierre Veen, Ed., Utrecht, The Netherlands, October 2000, pp. 87–93, STW Technology Foundation.

[16] Behrooz Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, New York, 2000.

[17] Jos T.J. van Eijndhoven, "16-bit compliant software IDCT on TriMedia/CPU64," Internal Report, Philips Research Laboratories, 1997.