# MPEG Macroblock Parsing and Pel Reconstruction on an FPGA-augmented TriMedia Processor

Mihai SIMA[†‡] Sorin COTOFANA[†] Stamatis VASSILIADIS[†]
Jos T.J. van EIJNDHOVEN[‡] Kees VISSERS[§]

[†]Delft University of Technology, Dept. of Electrical Engineering, Delft, The Netherlands
[‡]Philips Research – Dept. of Information and Software Technology, Eindhoven, The Netherlands
[§] TriMedia Technologies, Inc., Sunnyvale, California, U.S.A.
Phone: +31-(0)40-274-2593 E-mail: M.Sima@et.tudelft.nl

## Abstract

*This paper describes an experiment which aims to reveal the potential impact on performance yielded by augmenting a TriMedia-CPU64 processor with a multiple-context FPGA core. We first propose an extension of the TriMedia-CPU64 architecture, which consists of a Reconfigurable Functional Unit and its associated instructions. Then, we address the decoding of variable-length codes on such extended TriMedia and describe the architecture and FPGA-implementation of a Variable-Length Decoder (VLD) computing facility. When mapped on an ACEX EP1K100 FPGA, the proposed VLD exhibits a latency of* 7 *cycles. Preliminary results indicate that by configuring each of the VLD and 1-D IDCT (which is described elsewhere) facilities on a different FPGA context, and by activating the contexts as needed, the augmented TriMedia can perform macroblock parsing followed up by pel reconstruction with an improvement of* $20-25\%$ *over the standard TriMedia.*

## 1. Introduction

A common research question is the performance improvement that may be achieved by augmenting a general purpose processor with a reconfigurable core. The idea is to exploit both the general purpose processor flexibility to achieve medium performance for a large class of applications, and FPGA capability to implement application-specific computations. There have been various attempts to attach a reconfigurable core to a processor, most of them involving a simple processor [1, 2, 3]. This paper presents an experiment which aims to evaluate the potential impact on performance yielded by augmenting a TriMedia-CPU64 processor with a multiple-context FPGA core.

We first propose an extension of the TriMedia-CPU64 architecture, which encompasses a multiple-context FPGA-based Reconfigurable Functional Unit (RFU) and the associated instructions. With such extension, the user is given the freedom to define and use any computing facility subject to the FPGA size and TriMedia organization. In order to evaluate the potential of the proposed architectural extension, we chose a significant chunk of MPEG decoding as benchmark. In particular, we considered the parsing of Variable-Length (VL) coded data at the macroblock layer followed up by a pel reconstruction procedure. After developing a pure software implementation of this benchmark, we decided to provide FPGA hardware support for VL decoding and 1-D IDCT. By configuring each of the Variable-Length Decoder (VLD) and 1-D IDCT [12] facilities on a different FPGA context, and by activating the contexts as needed, the augmented TriMedia can compute macroblock parsing followed up by pel reconstruction with an improvement of $20-25\%$ over the standard TriMedia. Given the fact that TriMedia-CPU64 is a 5 issue-slot VLIW processor with 64-bit datapaths and a very rich multimedia instruction set [4], such an improvement within the target media processing domain indicates that the TriMedia + FPGA hybrid is a promising approach.

The paper is organized as follows. For background purposes, we briefly present several issues related to MPEG standard and the architecture of the FPGA core in Section 2. Section 3 describes the proposed architectural extension of TriMedia-CPU64. The VLD user-defined instruction, as well as pure software and FPGA-based implementation issues of the variable-length decoder are discussed in Section 4. The execution scenario of the chosen benchmark on both standard and extended TriMedia, and experimental results are presented in Section 5. Section 6 completes the paper with some conclusions and closing remarks.

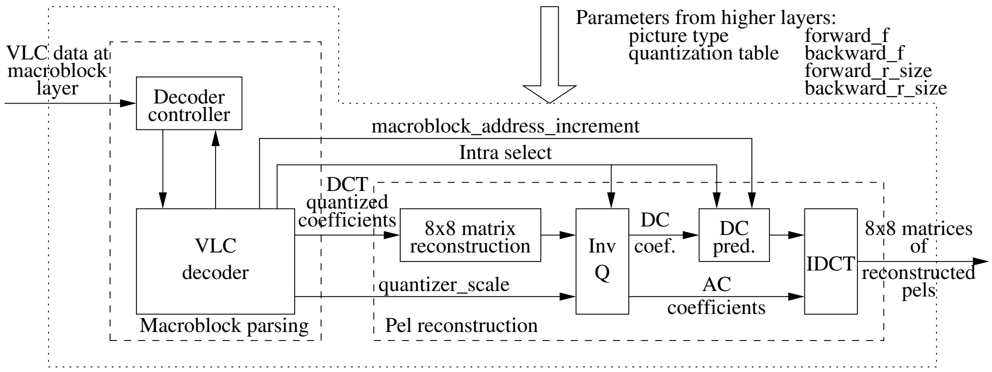


**Figure 1. Macroblock parsing and pel reconstruction module – adapted from [5].**

## 2. Background

To make the presentation self-consistent, we would like to address some issues related to macroblock parsing and pel reconstruction – two significant stages of MPEG decoding [5]. We also review the architecture of the FPGA that we used as an experimental reconfigurable core.

### 2.1. Macroblock parsing and pel reconstruction

The macroblock parsing process reads the VL coded data string from which all the headers corresponding to slice and higher layers have been removed, and outputs various symbols: *decoding parameters* at the macroblock layer (macroblock_address_increment, macroblock_type, coded_block_pattern, and quantizer_scale), *motion values*, and *composite symbols* (run/level pairs and end_of_block) which represent the DCT quantized coefficients. The decoding of the Variable-Length Codes (VLC) is performed according to a set of VLC tables defined by the MPEG standard. The motion values are used by a motion compensation process which is not considered here. However, since these values are decoded during the macroblock parsing, the overhead associated with the decoding of the motion values will be taken into consideration in the subsequent experiment.

Following the macroblock parsing, a pel reconstruction process recreates $8 \times 8$ matrices of pels. The pel reconstruction module is depicted in Figure 1. Its functionality is as follows. First, $8 \times 8$ matrices of DCT quantized coefficients are recreated by a Matrix Reconstruction module. Second, an inverse quantization (InvQ) is performed. An $8 \times 8$ quantization table, and a multiplicative quantization factor (quantizer_scale) are used in the InvQ process. Third, a DC prediction unit reconstructs the DC coefficient in intra-coded macroblocks. Finally, an IDCT is performed.

In connection with Figure 1 and the subsequent experiment, we would like to mention that the VLC decoder and IDCT will benefit from reconfigurable hardware support.

### 2.2. The multiple-context FPGA architecture

Field-Programmable Gate Arrays (FPGA) [6] are devices which can be configured *in the field* by the end user. In a general view, an FPGA is composed of two constituents: *Raw Hardware* and *Configuration Memory*. The function performed by the raw hardware is defined by the information stored into the configuration memory. Generally speaking, a multiple-context FPGA [7] is an FPGA having the configuration memory replicated in order to contain several configurations for the raw hardware. That is, a multiple-context FPGA contains an on-chip cache of raw hardware configurations, which are referred to as *contexts*. Such a cache allows a context switch to occur on the order of nanoseconds [9]. However, loading a new configuration from off-chip is still limited by low off-chip bandwidth.

In the sequel, we will assume a multiple-context FPGA which has the architecture of the raw hardware identical with that of an ACEX 1K device from Altera [8]. Our choice could allow future single-chip integration, as both ACEX 1K FPGAs and TriMedia are manufactured in the same TSMC technological process. Briefly, an ACEX 1K device contains an array of 4-input Look-Up Tables (LUT), a number of Embedded Array Blocks (EAB), each EAB being mainly a RAM block with 8 inputs and 16 outputs, and an interconnection network.

The next section will introduce the architectural extension for the TriMedia-CPU64.

## 3. An architectural extension for TriMedia

TriMedia-CPU64 is a 64-bit 5 issue-slot VLIW core [4], launching a long instruction every clock cycle. It has a uniform 64-bit wordsize through all functional units, the register file, load/store units, on-chip highway and external memory. Each of the five operations in a single instruction can (in principle) read two register arguments and

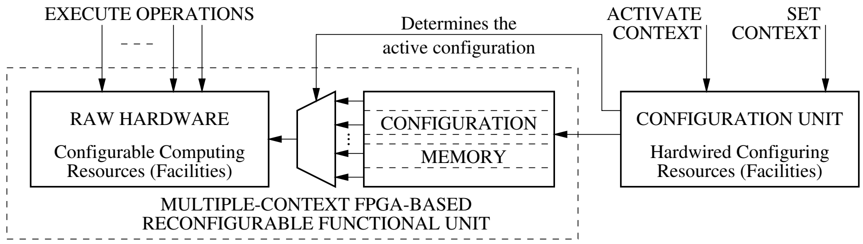


**Figure 2. The organization of the RFU and associated configuration unit.**

write one register result. The architecture supports subword parallelism and is optimized with respect to media processing. With the exception of floating point divide and square root, all functional units have a recovery of 1, while their latency ranges from 1 to 4. The TriMedia-CPU64 VLIW core also supports multi-slot operations, or super-operations. Such a super-operation occupies two neighboring slots in the VLIW instruction, and maps to a double-width functional unit. This way, operations with more than 2 arguments and/or more than one result are possible.

First we propose that the TriMedia-CPU64 processor is augmented with a Reconfigurable Functional Unit (RFU) which consists mainly of a multiple-context FPGA core. Also, a hardwired Configuration Unit which manages the reconfiguration of the raw hardware is associated to the reconfigurable functional unit, as it is depicted in Figure 2. 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 are required.

In order to use the RFU, a kernel of new instructions is needed. This kernel constitutes the extension of the TriMedia-CPU64 instruction set architecture we propose. Loading a context information into the RFU configuration memory is performed under the command of a `SET_CONTEXT` instruction, while the `ACTIVATE_CONTEXT` instruction controls the swaping of the active configuration with one of the idle on-chip configuration. `EXECUTE` instructions launch the operations performed by the computing resources configured on the raw hardware [10]. In this way, the execution of an RFU-mapped operation requires three basic stages: `SET_CONTEXT`, `ACTIVATE_CONTEXT`, and `EXECUTE`.

The user is given a number of `EXECUTE` instructions which encompass different operation patterns: single- or double-slot operations, operations with an immediate argument, etc. It is the responsibility of the user to choose the appropriate `EXECUTE` instruction corresponding to the pattern of the operation to be executed. At the source code level, this may be done setting up an *alias*, as it is described subsequently. Since the `EXECUTE` instructions are executed on the RFU without checking of the active configuration, it is still the responsibility of the user to perform the management of the active and idle configurations.

For the semantics of an operation performed by a computing facility, its latency, recovery, and slot assignment are all user definable, the source code of the application should contain information to augment the Machine Description File [11]. Assuming for example a user-defined `VLD` instruction, a way to specify such information is to annotate the source code as follows:

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

In a similar way, the user can define as many RFU-related instructions as she wants. The next section will present the sintax and semantics of the VLD instruction, as well as implementation issues of the VLD computing facility.

# 4. VLD instruction and computing facility

A VLD instruction which returns a single DCT symbol (*run/level* pair or *end-of-block*) per execution is considered. A super-operation pattern with two input (Rx, Ry) and two output (Rz, Rw) registers is assigned to the VLC decoder:

$$\texttt{VLD Rx, Ry} \rightarrow \texttt{Rz, Rw}$$

The Rx register specifies the decoding parameters which identify the type of the symbol to be detected: AC/DC, luminance/chrominance, intra/non-intra. The second register, Ry, contains 64 bits of the VL compressed data. The decoded symbol and its code length will be stored into registers Rz and Rw, respectively. In MPEG decoding, a new `VLD` operation can be launched only after the previous one has completed. Consequently, a recovery lower than the latency gives no advantages. Therefore, such implementation should not be sought.

**Table 1. The partitioning of the VLC codes of AC coefficients into groups and classes.**

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

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

### 4.1. VLD implementation on standard TriMedia

The implementation of the VLC decoder in the standard TriMedia is a modified version of that proposed in [13]. The VLD is implemented as a repeated table-lookup. Each lookup decodes a chunk of bits (8 bits at the first level lookup), and determines if a valid code was encountered. In case of a valid decode, a run-level pair is generated, or an escape or end-of-block flag is set. If a *miss* is detected, an offset into the VLC table and a chunk-size for a second-level lookup is generated. This process of signaling an incomplete decode and generating a new offset may be repeated three times. After compiling the C code and scheduling procedure, we evaluated that a table lookup takes 21 cycles. Consequently, the decoding of a single DCT coefficient can take between 21 and 63 cycles. The size of all lookup tables is 10 KB.

### 4.2. VLD implementation on FPGA

The VLD is implemented on FPGA as a parallel lookup into EABs, followed by a selection of the proper result. Since a single EAB can implement a lookup table of 8 inputs, we partitioned the VLC table according to this FPGA architectural characteristic, as presented in Table 1.

The implementation is presented in Figure 3. Regarding the groups 1, 2, and 3, a number of 1, 6, and 9 leading bits are shifted out from the *same* VLC string. The three new resulted strings are each sent to a different EAB, and three run/level pairs are generated as if the shifted leading bits would have been those mentioned in the column *Bypassed header*. By means of combinatorial circuits, the same procedure is carried out for groups 0, end-of-block, and escape.

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

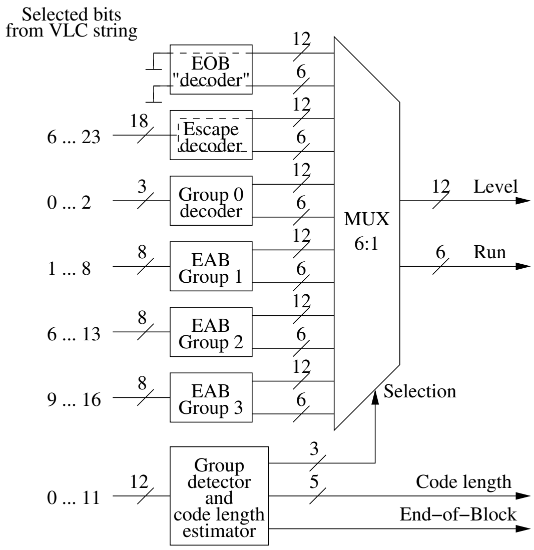


**Figure 3. The VLD implementation on FPGA.**

By simulation, we found that the FPGA-based VLD operation exhibits a latency of 7. After compiling and scheduling, we evaluated that a single DCT coefficient can be decoded in 11 cycles including all overheads. 6 EABs of an ACEX EP1K100 device are used.

# 5. Experimental results

In order to determine the potential impact on performance provided by the multiple-context reconfigurable core, we will consider a benchmark which consists of a macroblock parsing followed by pel reconstruction procedures. Therefore, we operate at MPEG slice level, i.e., the parameters of the slice and above layers are assumed to be constant. This computing scenario is presented in Figure 4. First, a variable-length decoding of a macroblock (header and DCT coefficients extraction) is performed. Then, the $8 \times 8$ matrices are recreated, and inverse quantization, followed by DC coefficient prediction for intra-coded macroblocks are carried out. After all macroblocks in a slice have been decoded, a burst of 2-D IDCTs is launched in order to reconstruct the initial matrices of pels.

A pure software implementation of the 2-D IDCT which can be scheduled into 56 cycles is claimed in [4], while an FPGA-based implementation exhibiting a throughput of one 2-D IDCT every 32 cycles is described in [12]. All the contexts of the RFU are to be configured at application load time, i.e., a number of `SET_CONTEXT` instructions are scheduled on the top of the program code. A sample of the code using the instructions of the architectural extension is presented subsequently. As it can be observed, the `VLD` and `IDCT` exhibit the same execution pattern: two inputs and two outputs.

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

Therefore, our experiment includes two approaches: *pure software* and *FPGA-based*. As mentioned, a DCT coefficient is decoded in 21-63 cycles, and a 2-D IDCT can be computed in 56 cycles in the pure software approach. In the FPGA-based approach, a DCT coefficient is decoded in 11 cycles, and the 2-D IDCT is carried out with the throughput of 1/32 IDCT/cycle. The context switching penality is 10 cycles.

## 5.1. Pel reconstruction performance evaluation

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

Subsequently, we present the results according to two scenarios: *worst-case*[1] and *average-case*. In both cases we assumed that an average of 5 non-zero coefficients per block are decoded. In the worst-case scenario, we assumed that all DCT coefficients produce a *hit* on the first level lookup when the pure software implementation is used. In the same worst-case scenario, we also assumed that the overhead introduced by parsing the macroblock headers has the largest value (for example, the quantization value is assumed to be updated every macroblock). Since the worst-case scenario coresponds to long variable-length codes, it is statistically not relevant. Therefore, we evaluated the performances in a average-case scenario. In such scenario, we assumed that two of five DCT coefficients produce a *miss* at the first lookup. Also, we weighted the overhead introduced by parsing the macroblock header with the transmiting probability of different decoding parameters of the macroblock layer. The results are presented in Table 2. The numbers indicate the improvements we get in connection with the number of cycles.

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

# 6. Conclusions and future work

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

[1] Considered from our point of view.

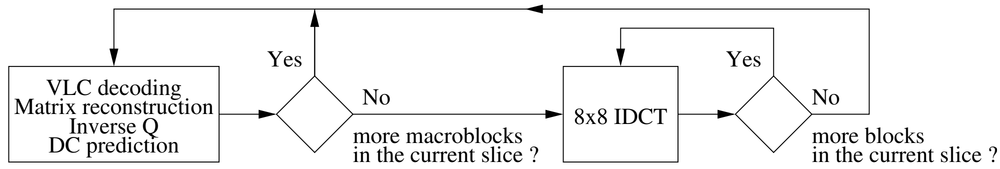


**Figure 4. The computing scenario of the macroblock parsing and pel reconstruction routine.**

**Table 2. Performance improvement of multiple-context FPGA-augmented TriMedia-CPU64 over "ideal-cache" (standard) TriMedia-CPU64 for a macroblock parsing followed by pel reconstruction application.**

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

## Acknowledgements

The authors would like to thank Evert-Jan Pol with Philips Research for numerous helpful comments.

## References

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

[2] Hauser, J.R., and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, California, 1997, pp. 12–21.

[3] Kastrup, B., A. Bink, and J. Hoogerbrugge, "ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator," in *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, California, 1999, pp. 92–100.

[4] van Eijndhoven, J.T.J., 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 *International Conference on Computer Design*, Austin, Texas, 1999, pp. 586–592.

[5] Mitchell, J.L., W.B. Pennebaker, C.E. Fogg, and D.J. LeGall, *MPEG Video Compression Standard*, Chapman & Hall, New York, New York, 1996.

[6] Brown, S., 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.

[7] DeHon, A., T. Knight Jr., E. Tau, M. Bolotski, I. Eslick, and D. Chen, "Dynamically Programmable Gate Array with Multiple Context," U.S. Patent No. 5,742,180, 1998.

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

[9] Trimberger, S., D. Carberry, A. Johnson, and J. Wong, "A Time-Multiplexed FPGA," in *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, California, 1997, pp. 22–28.

[10] Sima, M., S. Vassiliadis, S. Cotofana, J.T.J. van Eijndhoven, and K. Vissers, "A Taxonomy of Custom Computing Machines," in *PROGRESS Workshop on Embedded Systems*, Utrecht, The Netherlands, 2000, pp. 87–93.

[11] Pol, E.-J.D., 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 *International Conference on Computer Design*, Austin, Texas, 1999, pp. 593–598.

[12] Sima, M., S. Cotofana, J.T.J. van Eijndhoven, S. Vassiliadis, K. Vissers, "8 $\times$ 8 IDCT Implementation on an FPGA-augmented TriMedia," in *IEEE Symposium on FPGAs for Custom Computing Machines*, Rohnert Park, California, 2001.

[13] Pol, E.-J.D. "VLD Performance on TriMedia-CPU64," Private Communication, Philips Research, Eindhoven, The Netherlands, 2000.