Inverse Quantization on FPGA-augmented TriMedia

Mihai Sima[†‡] Stamatis Vassiliadis[†] Sorin Cotofana[†] Jos T.J. van Eijndhoven[‡]
[†]*Delft University of Technology, Dept. of Electrical Engineering, Delft, The Netherlands*
[‡]*Philips Research, Dept. of Information and Software Technology, Eindhoven, The Netherlands*
M.Sima@ewi.tudelft.nl http://ce.et.tudelft.nl/~mihai

Abstract—**This paper investigates inverse quantization on FPGA-augmented TriMedia processor. First, we outline the extension of TriMedia architecture consisting of FPGA-based Reconfigurable Functional Units (RFU) and associated generic instructions. Then we analyse an IQ-4 (RFU–specific) instruction which can process four coefficients per call, and propose a scheme to implement the IQ-4 operation on the RFU. When mapped on an ACEX EP1K100 FPGA, the proposed IQ-4 exhibits a latency of 18 and a recovery of 2 TriMedia-32@200 MHz cycles, and occupies 43% of the device. By configuring the IQ-4 facility on the RFU at application load-time, inverse quantization can be computed on FPGA-augmented TriMedia with a speed-up of $1.5\times$ over the standard TriMedia.**

Keywords—**Reconfigurable computing; inverse quantization; VLIW processors; field-programmable gate arrays.**

I. Introduction

Enhancing a general purpose processor with a reconfigurable core is a common issue addressed by computer architects [5], [12], [2]. The idea is to exploit both the processor flexibility to achieve medium performance for a large class of applications, and FPGA capability to implement application-specific computations. An instance of such enhanced processor is TriMedia+FPGA hybrid [7], which proved promising results with respect to several applications: Inverse Discrete Cosine Transform [6], Entropy Decoding [8], and $Y'CbCr$-to-$R'G'B'$ Converter [9].

Inverse Quantization is a computing-intensive stage of MPEG decoding. Traditionally, IQ has been captured directly in customized hardware in Application-Specific Instruction Processors, or carried out in software in media-domain processors. In this paper, we describe a reconfigurable IQ design for FPGA-augmented TriMedia, and demonstrate that significant speed-up can be achieved over standard TriMedia for an IQ application.

Since Inverse Quantization exhibits large data and instruction-level parallelisms, it can be implemented on standard TriMedia with high efficiency. Obtaining improvements for a task having a computational pattern which TriMedia has been optimised for, is indeed challenging. The main idea in achieving speed-up is to configure on FPGA a pipelined inverse quantizer and to build a software pipeline routine calling this FPGA-mapped unit. In particular, we provide reconfigurable-hardware support for an IQ-4 operation which can process four coefficients per call. When mapped on an ACEX EP1K100 FPGA, the computing unit performing the IQ-4 has a latency of 18 and recovery of 2 TriMedia@200 MHz cycles, and occupies 43% of the device.

The experimental results indicate that by configuring the IQ-4 unit on FPGA at application load-time, inverse quantization can be computed on FPGA-augmented TriMedia $1.5\times$ faster over the standard TriMedia. Given the fact that the experimental TriMedia is a 5 issue-slot 64-bit media-oriented VLIW processor [10], such an improvement within the target media processing domain indicates that TriMedia+FPGA hybrid is a promising approach.

Summarizing, the paper contributions are:

- The syntax and the semantics of the IQ-4 user-defined operation.
- The IQ-4 computing unit implementation on an ACEX EP1K100 FPGA from Altera.
- A high performance inverse quantization implementation on FPGA-augmented TriMedia.

The paper is organized as follows. We present several issues related to inverse quantization in Section II. Section III outlines the architectural extension of the TriMedia processor. Several considerations regarding a pure-software solution are made in Section IV The FPGA-based implementation of the IQ-4 unit which dequantizes four coefficients per call is discussed in Section V. The experimental framework and the results are presented in Section VI. Section VII concludes the paper.

II. Background

Quantization is basically a process for reducing the precision of the DCT coefficients. Precision reduction is extremely important, since lower precision almost always implies a lower bit rate in the compressed data stream.

The quantization process involves division of the integer DCT coefficient values by integer quantizing values. The result is an integer and fraction, and the fractional part must be rounded according to the rules defined by MPEG. It is the quantized values that is transmitted to the decoder.

For reconstruction, the decoder must first *dequantize* the quantized DCT coefficients, to reproduce the DCT coefficients computed by the encoder. Essentially, the Inverse Quantization (IQ) algorithm scales every element by a unique quantized weight. Since some precision was lost in quantizing, the reconstructed DCT coefficients are necessarily approximations to the values before quantization.

After entropy decoding, the two-dimensional array of coefficients, $QF[v][u]$, is inverse quantised to produce the reconstructed DCT coefficients, $F[v][u]$. In MPEG2, Inverse Quantisation (IQ) consists of three stages: Inverse Quantisation Arithmetic, Saturation, and Mismatch Control [3]. The inverse quantisation arithmetic produces $F''[v][u]$ coefficients. For DC coefficients in intra-coded blocks, Equation 1 is used:

$$F''[0][0] = \text{intra\_dc\_mult} \times \text{QF}[0][0] \tag{1}$$

where the factor *intra_dc_mult* is derived from the data element *intra_dc_precision* according to Table 7-4 of the ITU-T Recommendation H.262 [3]. Basically, Equation 1 specifies a scaling-up by a factor of 8, 4, 2, or 1. For all other coefficients, the following equation should be used:

$$F''[v][u] = (2 \times QF[v][u] + k) \times W[w][v][u] \times \\ \times \text{quantizer\_scale}/32 \tag{2}$$

where

$$k = \begin{cases} 0 & \text{intra blocks} \\ \text{sign(QF[v][u])} & \text{non-intra blocks} \end{cases} \tag{3}$$

The factor *quantizer_scale* is an unsigned integer and is encoded as a 7-bit fixed-length code. Thus, it has values in the range $\{1, \ldots, 31\}$, inclusive (0 is not allowed). Each weighting coefficient, $W[w][v][u], w = 0 \ldots 3, v = 0 \ldots 7, u = 0 \ldots 7$, is represented on an 8-bit unsigned integer, and extracted during the parsing of the sequence header. The operator /' represents the integer division with truncation of the result toward zero.

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range $[-2048 \cdots + 2047]$. Finally, the mismatch control operation toggles the least significant bit of $F[7][7]$ if the double sum $\sum_{v=0}^{7} \sum_{u=0}^{7} F[v][u]$ of all DCT coefficients is even.

We would like to mention that MPEG defines rules for changing the quantization of the DCT coefficients from place to place in the image as follows. The factor *quantizer_scale* is derived from the data elements *quantizer_scale_code* and *quantizer_scale_type* according to Table 7-6 of the ITU-T Recommendation H.262 [3], and therefore can be changed per coded macroblock. However, the factor *intra_dc_mult* can be changed only per picture. Since we use only MP@ML MPEG conformance bit-strings in all subsequent experiments, only two weighting matrices (one for intra-coded blocks, and the other for non-intra-coded blocks) are used for inverse quantization. Thus, $w = \{0, 1\}$.

For the inverse quantization, all the mentioned values should be regarded as parameters. Consequently, the inverse quantization routine has to read in both the DCT coefficients to be dequantized and the following parameters: the weighting array W, the *quantizer_scale*, and an intra/non-intra flag.

Before we present the FPGA–based implementation of the IQ-4 computing facility, we outline the architectural extension of the TriMedia processor.

III. Architectural extension for TriMedia

TriMedia–CPU64 is a processor which features a rich instruction set optimized for media processing. Specifically, it is a 5 issue-slot 64-bit VLIW engine, launching a long instruction every clock cycle [4]. Each of the five operations in a single VLIW instruction can in principle read two register arguments and write one register result. The processor also supports double-slot operations, or super-operations [11]. 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 two arguments and one result are possible. The architecture supports subword parallelism: for example, operations on 8-bit unsigned integer vectors, or on 16-bit signed integer vectors are possible.

Following the methodology described in [6], [8], TriMedia can be augmented with an FPGA-based Reconfigurable Functional Unit (RFU). The RFU 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. Even though only double-slot operations are supported by the current TriMedia simulator, we propose to extend the concept of super-operations and provide RFU on which up to 5-slot operations can be executed. This extension will be very useful when vectorial operations are mapped on the configurable hardware.

In order to use an RFU, new instructions are provided: `SET`, and `EXECUTE`. Loading a new configuration into an RFU is controlled by a `SET` instruction, while `EXECUTE` (generic) instructions launch the operations performed by the computing resources configured on the FPGA. 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. For more details regarding this issue we refer the reader to bibliography [7].

Several considerations about the latency of an RFU-configured computing resource are worth to be provided. Due to layout constraints, the RFU is likely to be located far away from the Register File (RF) in the floorplan of the TriMedia. The immediate effect is that there will be large delays in transferring data between the RFU and RF. Consequently, *read* and *write back* cycles have explicitely to be provided. In such circumstances, the latency of an RFU-based computing resource is composed of 1 cycle for *read*, the number of cycles corresponding to the FPGA delay, and 1 cycle for *write back*.

Next, a pure-software implementation of inverse quantization is discussed.

IV. IQ PURE-SOFTWARE IMPLEMENTATION

After variable-length decoding, each DCT coefficient is represented on a 16-bit signed integer. Thus, the 8×8 matrix can be thought as being stored in 16 four-element vectors. In this way, the IQ implementation can intensively use four-way SIMD operations.

In the pure-software solution, all 64 coefficients are first inverse quantised with the general Formula 2, and then saturated. In parallel, the intra DC coefficient is scaled-up according to Equation 1. Next, if the block is intra-coded, the top left-handed DCT coefficient of the 8×8 block is replaced with this DC coefficient. Finally, mismatch sum is computed and the least significant bit of $F[7][7]$ is updated accordingly. We would like to mention that a separate IQ routine has been designed for dequantizing each of the intra-coded and non-intra-coded information. The rationale behind this strategy is to bypass the computation of the **signum** function of $QF[7][7]$, and also the addition of the term $k \equiv 0$ for intra-coded blocks.

After developing C-level code that makes intensively use of TriMedia–CPU64 custom operations, compiling it, and running the executable on a cycle-accurate simulator, we determined that an 8×8 matrix can be dequantized in 39 cycles for intra-coded blocks, and 52 cycles for non-intra-coded blocks (`LOAD` and `STORE` operations are taken into account). 26 `NOP`s are inserted by the TriMedia scheduler in the 39-cycle routine for intra-coded blocks, which translates to an average utilization of 4.33 out of 5 operations per VLIW instruction. For non-intra-coded blocks, 30 `NOP`s are inserted in the 52-cycle routine, which means that 4.41 out of 5 operations are issued per instruction. Since the average utilization of the issue slots reaches such a large value, we can state that the TriMedia–CPU64 runs close to its full processing speed, and the pure-software IQ implementation on TriMedia–CPU64 constitutes a real challenge for an FPGA-based solution.

In inverse quantization of an 8×8 block, each and every pixel but the bottom right-hand one (which is subject to the mismatch operation) is dequantized independently of any other pixel in the block. Thus, IQ is mostly a feed-forward task that exhibits a large data-level parallelism. Consequently, the entire IQ computation can benefit from reconfigurable support if sufficient reconfigurable hardware is available. This way, the VLIW core will have only to load new data from and write the computed data back to main memory. In the next section, a number of details regarding IQ implementation on FPGA are outlined.

V. IQ IMPLEMENTATION ON FPGA

As mentioned, the number of pixels that can simultaneously be inverse quantized on FPGA is subject to the raw hardware logic capacity. On an ACEX EP1K100 FPGA, we succeeded to map an IQ-4 unit that can process four coefficients per call. This way, a burst of sixteen IQ-4 operations has to be launched in order to dequantize an entire 8×8 block. As depicted in Figure 1, the IQ-4 circuitry is structured as follows: the first part implements the IQ arithmetic (which is defined by Equations 1 and 2) and subsequent saturation, while the last part is a finite state machine implementing the mismatch control operation.

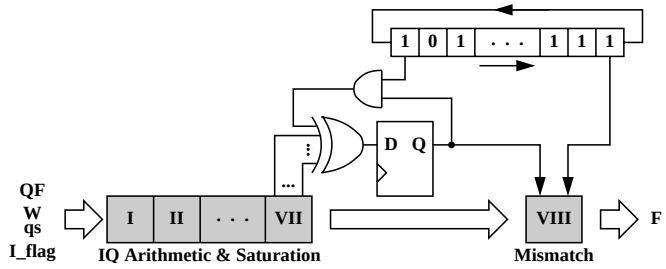

Fig. 1. **The IQ-4 implementation on FPGA.**

The reduction modules corresponding to multiplications by $W[v][u]$ (8-bit unsigned integer) and $\mathrm{quantizer\_scale}$ (7-bit unsigned integer) have been splitted-up in order to fit into an 100 MHz pipeline. No special optimization technics to reduce the partial product matrices have been employed; instead, we rely on the FPGA mapping tools in detecting *carry-propagate* (which is fast on FPGA) primitive. The factors **intra_dc_mult** and **quantizer_scale** are generated inside FPGA from the MPEG data elements **intra_dc_precision**, respectively **quantizer_scale_code** and **q_scale_type**.

In addition to the feed-forward circuitry for IQ arithmetic and saturation computation, the IQ unit also includes a finite state machine that co ntrols the processing of the DC component in intra-coded blocks, as well as the mismatch operation as follows:

• During the first out of sixteen IQ-4 calls needed for processing an 8×8 block, the fourth element of the QF[3..0] vector (i.e., the DC component) is dequantized according to Equation 1 for intra-coded blocks, and Equation 2 for non-intra-coded blocks.
• The mismatch information is accumulated during sixteen successive IQ-4 calls, and updates the last DCT coefficient accordingly at the end of each 16^{th} call.
Thus, the IQ-4 unit we propose is a circuitry with state (non-re-entrant functional unit). In order to ensure a correct response, a block should be completely processed before a new one is being considered. Furthermore, the 64-bit word containing the DC component should be processed firstly, and the 64-bit word containing the highest spatial frequency component should be processed lastly.

By writing and synthesizing VHDL code, we determined that 8 pipeline stages are needed to implement the IQ-4 unit on an ACEX EP1K100 FPGA, which translates into a latency of $8 \times 2 + 1 + 1 = 18$ and a recovery of 2 TriMedia@200 MHz cycles. It worth to mention that IQ-4 unit occupies 43% of the logic cells, and 171 out of 333 I/O pins of the mentioned reconfigurable device.

We would also like to mention that, on the same device, we did not succeed to map an IQ-8 unit that processes eight coefficients per call. Although about 80% of the logic cells of the ACEX EP1K100 array would be occupied by the IQ-8 unit, the FPGA mapping tools did not succeed to map the circuitry mainly due to the large numbers of I/O pins that are needed. Indeed, 331 out of 333 I/O pins would be used by IQ-8. The pin limitation in FPGA-based circuitry is a known problem – see for example [1]. A way to overcome this limitation is to provide for a larger FPGA having more I/O pins (and, implicitly, more raw hardware). However, this solution is more expensive in terms of silicon area for the same number of I/O pins, since the logic capacity increases with the square root of the chip edge, while the number of I/O pins increases only linearly with the chip edge. A second solution is to emulate an IQ-8 unit processing two 8×8 blocks by two IQ-4 units each mapped on a smaller RFU, and each processing a separate 8×8 block.

In the next section, we present a routine that contain calls to FPGA-mapped IQ computing unit, and compare the performance achieved on FPGA-augmented TriMedia over the standard TriMedia.

VI. Experimental results

Since the FPGA-mapped IQ is a circuitry with state, two operations are needed to control the unit: one that resets the finite state machine, and the other that launches the proper IQ operation. Assuming an IQ-4 unit, the syntax of each operation is:

EXECUTE <RESET-IQ-4> →

EXECUTE <IQ-4> R_QF, R_W, R_qs, R_param → R_F

The first operation has a latency of 3 cycles, while, as mentioned, the later (2-slot) operation has a latency of 18 cycles and a recovery of 2 cycles. For reasons that will become relevant later on, the inverse quantization is carried out at slice level. That is, the entire slice is inverse quantized by means of `EXECUTE` $<\mathbf{IQ-4}>$ instructions before a reconfiguration of the RFU to implement a different function may be considered.

To inverse quantize an 8×8 block of coefficients, sixteen `IQ-4` operations are launched in a row. Before and after the RFU calls, `LOAD` and `STORE` operations fetch the input operands from main memory into register file, and store the results back into memory, respectively. Since the code is very simple and symmetrical, generating a tight software-pipeline loop by programming directly in assembly is indeed feasible, as depicted in Figure 2. As it can be observed, the loop is folded at Cycles 4 and 35, thus a throughput of 1/32 IQ/cycle is achieved. The first two `LOAD` operations that are executed during the previous loop iteration, and the last 9 `STORE` operations that are executed during the next loop iterations generate an overhead for firing-up and flushing the software pipeline of 24 cycles. In addition, loading the $W[w][v][u], w = 0 \ldots 1, v = 0 \ldots 7, u = 0 \ldots 7$ array from memory into register file needs 16 `LOAD` operations, that is, 8 cycles. Thus, the total overhead for firing-up and flushing the software pipeline is 32 cycles.

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. 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 into the worst case penality associated to the prologue and epilogue of the software pipeline loop of $32/38 \approx 0.84$ cycles/block. Since this overhead represents about 2.5% of the 32 cycle/block throughput in the most disadvantageous case, it can be neglected.

Thus, the global occupancy figure is 74 out of $32 \times 5 = 160$, which means that 2.31 out of 5 issue-slots are filled in with operations. There are plenty of free slots that can be utilized for other purposes, e.g., implementing an additional pure-software IQ. Thus, the throughput figure of 1/32 blocks/cycle represents the lower bound of the performance improvement.

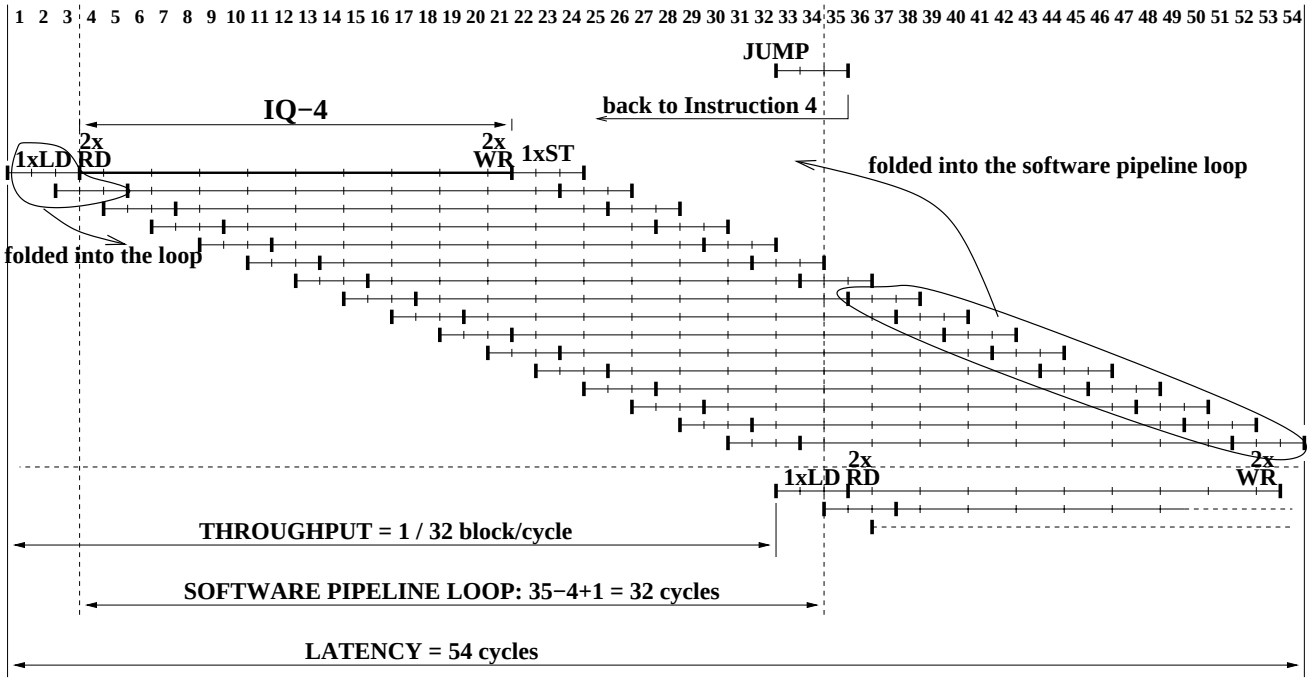

Fig. 2. **Schedule result for the IQ-4 unit (LD stands for `LOAD`, RD for read, WR for write, and ST for `STORE`).**

VII. Conclusions and Future Work

We have described an inverse quantizer on FPGA-augmented TriMedia. For such a task, the performance improvement over the standard TriMedia is approx. 50% in terms of speed. The major lesson learned is that deep pipelines implemented on the RFU can provide significant improvements even for a performant VLIW processor within its target media domain.

References

[1] Jonathan Babb, Russell Tessier, and Anant Agarwal. Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators. In *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM '93)*, pages 142–151, Napa Valley, California, April 1993.

[2] John R. Hauser and John Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *5th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '97)*, pages 12–21, Napa Valley, California, April 1997.

[3] International Telecommunication Unit. Information technology – Generic coding of moving pictures and associated audio information: Video. ITU-T Recommendation H.262, February 2000.

[4] Selliah Rathnam and Gerrit A. Slavenburg. An Architectural Overview of the Programmable Multimedia Processor, TM-1. In *Forty-First IEEE Computer Society International Conference: Technologies for the Information Superhighway (COMPCON96)*, pages 319–326, Santa Clara, California, February 1996.

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

[6] Mihai Sima, Sorin Cotofana, Jos T.J. van Eijndhoven, Stamatis Vassiliadis, and Kees Vissers. 8×8 IDCT Implementation on an FPGA-augmented TriMedia. In *9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, page n/a, Rohnert Park, California, April 2001.

[7] Mihai Sima, Sorin Cotofana, Stamatis Vassiliadis, Jos T.J. van Eijndhoven, and Kees Vissers. MPEG Macroblock Parsing and Pel Reconstruction on an FPGA-augmented TriMedia Processor. In *IEEE International Conference on Computer Design*, pages 425–430, Austin, Texas, September 2001.

[8] Mihai Sima, Sorin D. Cotofana, Stamatis Vassiliadis, Jos T.J. van Eijndhoven, and Kees A. Vissers. MPEG-compliant Entropy Decoding on FPGA-augmented TriMedia/CPU64. In *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2002)*, pages 261–270, Napa Valley, California, April 2002.

[9] Mihai Sima, Stamatis Vassiliadis, Sorin Cotofana, and Jos T.J. van Eijndhoven. Color Space Conversion for MPEG decoding on FPGA-augmented TriMedia Processor. In *IEEE 14th International Conference on Application-specific Systems, Architectures, and Processors (ASAP 2003)*, pages 250–259, The Hague, The Netherlands, June 2003.

[10] Jos T. J. van Eijndhoven, F. W. Sijstermans, Kees A. Vissers, E.-J. D. Pol, M. J. A. Tromp, P. Struik, R. H. J. Bloks, Pieter van der Wolf, A. D. Pimentel, and Harald P.E. Vranken. TriMedia CPU64 Architecture. In *Proceedings of International Conference on Computer Design*, pages 586–592, Austin, Texas, October 1999.

[11] Jos T.J. van Eijndvhoven, Gerrit A. Slavenburg, and Selliah Rathnam. VLIW Processor has different functional units operating on commands of different widths. U.S. Patent No. 6,076,154, June 2000.

[12] Ralph D. Wittig and Paul Chow. OneChip: An FPGA Processor With Reconfigurable Logic. In *4th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, pages 126–135, Napa Valley, California, April 1996.