Color Space Conversion for MPEG decoding on FPGA-augmented TriMedia Processor

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

Abstract

A case study on Color Space Conversion (CSC) for MPEG decoding, carried out on FPGA-augmented TriMedia processor is presented. That is, a transform from $Y'CbCr$ color space to $R'G'B'$ color space is addressed. First, we outline the extension of TriMedia architecture consisting of FPGA-based Reconfigurable Functional Units (RFU) and associated instructions. Then we analyse a CSC (RFU–specific) instruction which can process four pixels per call, and propose a scheme to implement the CSC operation on RFU(s). When mapped on an ACEX EP1K100 FPGA, the proposed CSC exhibits a latency of 10 and a recovery of 2 TriMedia@200 MHz cycles, and occupies 57% of the device. By configuring the CSC facility on the RFU(s) at application load-time, color space conversion can be computed on FPGA-augmented TriMedia with 40% speed-up over the standard TriMedia.

1. Introduction

Enhancing a general purpose processor with a reconfigurable core is a common issue addressed by computer architects [8, 15, 2]. The basic idea of this approach 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 [11], on which the user is given the freedom to define and use any computing facility subject to FPGA size and TriMedia organization. Several applications implemented on this hybrid, e.g., Inverse Discrete Cosine Transform [10] and Entropy Decoding [12], proved promising results. In this paper, we address color space conversion which is carried out at the end of the MPEG decoding process.

The last stage of the MPEG decoding consists of a color space conversion, which is a linear transform from $Y'CbCr$ color space to $R'G'B'$ color space. Since this transform exhibits large data and instruction-level parallelisms, it can be implemented on TriMedia with very high efficiency. Obtaining improvements for a task having a computational pattern which TriMedia has been optimised for, is indeed challenging.

In this paper we demonstrate that significant speed-up for $Y'CbCr$-to-$R'G'B'$ color space conversion can be achieved on FPGA-enhanced TriMedia over standard TriMedia. The main idea is to configure a pipelined Color Space Converter (CSC) on FPGA and to unroll the software loop issuing an CSC operation such that the penalty associated to firing-up and flushing the CSC pipeline is reduced. In particular, we provide configurable-hardware support for a CSC operation which can process four pixels per call. When mapped on an ACEX EP1K100 FPGA from Altera, the computing unit performing the CSC operation has a latency of 10 and recovery of 2 TriMedia@200 MHz cycles, and occupies 57% of the device. The simulations carried out on a TriMedia cycle accurate simulator indicate that by configuring the CSC unit on FPGA at application load-time, $Y'CbCr$-to-$R'G'B'$ color space conversion can be computed on extended TriMedia 40% faster over the standard TriMedia. Given the fact that the experimental TriMedia is a 5 issue-slot 64-bit VLIW processor with a very rich multimedia instruction set [14], such an improvement within the target media processing domain indicates that the TriMedia + FPGA hybrid is a promising approach with respect to color space conversion.

Summarizing, the paper contributions are:

- The syntax and the semantics of the CSC user-defined operation.
- The CSC computing facility implementation on an ACEX EP1K100 FPGA from Altera.
- A high performance implementation of color space conversion on FPGA-augmented TriMedia.

The paper is organized as follows. We present several issues related to color space conversion in Section 2. Section 3 outlines the TriMedia architectural extension that incorporates the reconfigurable array. The FPGA-based implementation of a color space converter and its associated instructions are discussed in Section 4. The experimental framework and the results are presented in Section 5. Section 6 concludes the paper.

2. Background

MPEG is a digital compression standard for multimedia [6, 1]. As depicted in Figure 1, the last stage of the MPEG decoding process is an $Y'CbCr$-to-$R'G'B'$ color space conversion. That is, gamma-corrected red, green, blue (R', G', and B') are computed from a luminance-related quantity, Y', and two color-related quantities, Cb and Cr.

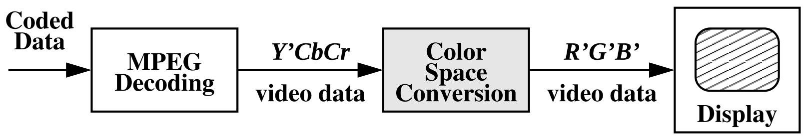

Figure 1. The video decoding process

To make the presentation self-consistent, we will address some issues related to color space conversion, which is carried out at the end of the MPEG-2 [5] decoding process.

2.1. Color space conversion

According to the Trichromatic Theory, it is possible to match all of the colors in the visible spectrum by appropriate mixing of three primary colors. Which primary colors are used is not important as long as mixing two of them does not produce the third. For display systems that emit light, the Red-Green-Blue (RGB) primary system is used.

A color space is a mathematical representation of a set of colors. In the sequel, we will present two color spaces: $R'G'B'$ and $Y'CbCr$.

2.1.1. $R'G'B'$ color space

Film, video, and computer-generated imagery all start with red, green, and blue intensity components. In video and computer graphics, the nonlinearity of the CRT monitor is compensated by applying a nonlinear transfer function to RGB intensities to form *Gamma–Corrected Red*, *Green*, and *Blue* ($R'G'B'$). The gamma-corrected red, green, blue are defined on a relative scale from 0 to 1.0, chosen such that shades of gray are produced when $E'_R = E'_G = E'_B$, where $E'_.$ denotes the analog gamma–pre-corrected signal associated with the primary X color.

In digital video, the analog signal is uniformly-quantized on 8 bits, so that 256 equally spaced quantization levels are specified. Coding range in computing has a *de facto* standard excursion, 0 to 255. Studio video provides footroom below the black code, and headroom above the white code; its range is standardized from 16 to 235. However, values less than 16 and greater than 235 are allowed in order to accomodate the transients that result from filtering.

2.1.2. $Y'CbCr$ color space

The data capacity accorded to color information in a video signal can be reduced as follows. First, $R'G'B'$ is transformed into luminance-related quantity called *luma* (Y'), and two color difference components called *chroma* (Cb, Cr) [7]. Since the human visual system has poor color acuity, the color detail can then be reduced by subsampling (lowpass filtering) without the viewer noticing.

The $Y'CbCr$ color space was developed as part of ITU-R Recommendation BT.601 [4]. All components are represented as 8-bit unsigned integers. Y' is defined to have a nominal range of 16 to 235; Cb and Cr are defined to have a range of 16 to 240, with 128 equal to zero. It is Y', Cb, Cr values that are coded inside an MPEG string. It is worth mentioning that Y', Cb, Cr are represented on 16-bit signed integers after motion compensation.

2.1.3. $Y'CbCr$-to-$R'G'B'$ conversion

If the gamma-corrected RGB data has a range of 0 to 255, as is commonly found in computer systems, the following equations describe the $R'G'B'$-to-$Y'CbCr$ conversion:

$$\begin{cases} R' = 1.164(Y' - 16) + 1.596(Cr - 128) \\ G' = 1.164(Y' - 16) - 0.813(Cr - 128) \\ \qquad\qquad\qquad\qquad - 0.391(Cb - 128) \\ B' = 1.164(Y' - 16) + 2.018(Cb - 128) \end{cases} \tag{1}$$

Even though Y' is defined to have a range of 16 to 235, while Cb and Cr have a range of 16 to 240, sample values outside the above mentioned ranges may occasionally occur at the output of the MPEG-2 decoding process according to ITU-T Recommendation H.262 [5]. As a consequence, $R'G'B'$ values must be saturated at the 0 and 255 levels after conversion to prevent any overflow and underflow errors.

With connection to the subsequent experiment, we would like to mention that the mapping defined by Equation set 1 will benefit from configurable hardware support.

2.2. $Y'CbCr$ sampling format conversion

As mentioned, since the eye is less sensitive to color information than brightness, the chroma channels can have a lower sampling rate that the luma channel without a dramatic degradation of the perceptual quality. In MPEG, 2:1 horizontal downsampling with 2:1 vertical downsampling is employed. That is, the Cb and Cr pixels lie between the Y' pixels on every other pixel on both the horizontal and vertical lines. Thus, a two-dimensional 2-fold upsampling has to be carried out before the proper color space conversion.

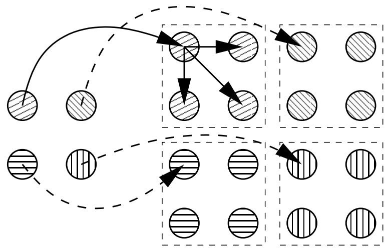

Figure 2. Two-dimensional 2–fold upsampling by replication

The simplest upsampling method employs a zero-order hold. That is, each and every pixel is replicated to East, South-East, and South, as depicted in Figure 2. Consequently, no additional processing is needed for upsampling at the expense of a poor frequency response characteristic. Indeed, since the zero-order hold does not possess a sharp cutoff frequency response characteristic, being a poor *anti-image* filter, it passes undesirable image components [13].

The filtering process is beyond the paper scope. Thus, we will consider that a zero-order hold is used, although this solution does not provide satisfactory image quality. Synthesizing, the following stages has to be performed in the process of color space conversion carried out at the end of MPEG decoding:

1. *Chroma* upsampling;
2. $Y'CbCr$-to-$R'G'B'$ linear transform.

Before we will present the FPGA–based implementation of the CSC computing facility and its associated user-defined instruction, we will outline the architectural extension of the experimental TriMedia processor.

3. Architectural extension for TriMedia

From the TriMedia family, TriMedia/CPU64 will be used as an experimental platform in all the subsequent tests. TriMedia/CPU64 is a processor model whose architecture features a rich instruction set optimized for media processing. Specifically, it is a 5 issue-slot 64-bit VLIW core, launching a long instruction every clock cycle [14]. 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 VLIW instruction can in principle read two register arguments and write one register result. The processor also supports 2-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 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 [10, 12], TriMedia processor can be augmented with one or more FPGA-based Reconfigurable Functional Units (RFU). An 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 2-slot operations are supported by the current TriMedia simulator, we propose to extend the concept of super-operations and provide RFUs 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 performed under the command of a `SET` instruction, while `EXECUTE` (generic) instructions launch the operations performed by the computing resources configured on the FPGA. With such 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 [11].

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, and the RFU will not benefit from bypassing capabilities of the RF [14]. 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*.

For the subsequent experiment, two instances of the TriMedia+FPGA hybrid are considered:

1. TriMedia @ 200MHz augmented with a single RFU, which can run at maximum one half of TriMedia clock frequency, that is, 100 MHz.
2. TriMedia @ 200MHz augmented with two RFUs, each running at maximum one quarter of TriMedia clock frequency, that is, 50 MHz.

In the sequel, the FPGA–based implementation of a CSC computing unit, and its associated instruction are presented.

4. CSC custom instruction and computing unit

Since three values (red, green, and blue) are to be computed for each pixel, we propose to provide configurable-hardware support for a 3-slot CSC operation which reads the $Y'CbCr$ triplet and returns the $R'G'B'$ triplet:

$$\textbf{CSC} \quad \boldsymbol{Y', Cb, Cr \longrightarrow R', G', B'}$$

where Y', Cb, Cr, R', G', B' are all 64-bit registers. Subject of the FPGA logic capacity and the number of FPGA I/O pins, a different number of pixels can be processed in parallel. Given the fact that the *luma* and *chroma* are represented as 16-bit signed integers, and gamma-corrected red, green, and blue are represented as 8-bit unsigned integers, at most four pixels can be processed in parallel. Indeed, the CSC is a 4-way SIMD operation which transforms three 16-bit signed integer vectors (Y', Cb, Cr) into three 8-bit unsigned integer vectors (R', G', B'). This translates to a number of $3 \times 4 \times 16 + 3 \times 4 \times 8 = 288$ I/O pins, which is acceptable for most FPGAs in general, and ACEX EP1K100 device in particular.

Since the current TriMedia simulator does not support super-operations on more than 2 slots, our 3-slot CSC operation has to be emulated by sequences of 1- and/or 2-slot operations. Therefore, we define two 2-slot CSC instructions: CSC_R, which performs the proper color space conversion and returns only the *red* information, and CSC_GB, which returns the *green* and *blue* information:

$$\begin{aligned} \textbf{CSC\_R} \quad & \boldsymbol{Y', Cb, Cr \longrightarrow R'} \\ \textbf{CSC\_GB} \quad & \boldsymbol{\longrightarrow G', B'} \end{aligned}$$

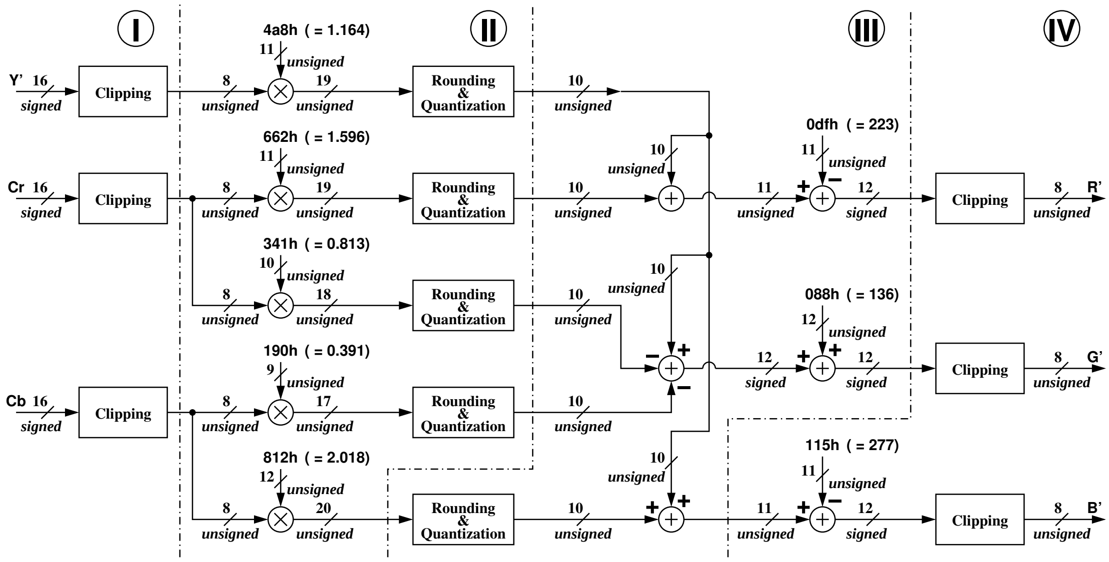

Figure 3. The CSC implementation on FPGA (the Roman numerals indicate the pipeline stages)

We have to emphasize that this approach is carried out only for experimental purpose. Fortunately, our choice does not generate overhead, since it is easier to schedule a single 3-slot instruction than multiple 1- and/or 2-slot instructions.

The FPGA–based CSC implementing the Equation set 1, is presented in Figure 3. By writing RTL-level VHDL code, we succeeded to identify a four-stage pipelined implementation which can run at 100 MHz on ACEX EP1K100 device. Adding the penalty of the extra read and write back cycles for an RFU–based operation, the CSC has a latency of 10 and recovery of 2 TriMedia@200MHz cycles if an RFU@100MHz is considered. For an RFU@50MHz, two pipelines stages can be merged into one, which translates into a CSC having the latency of 10 and recovery of 4.

5. Experimental results

For the first extended TriMedia instance, a CSC computing unit having the latency of 10 and recovery of 2 cycles is configured on the RFU@100MHz, while a CSC computing unit with the latency of 10 and recovery of 4 cycles is configured on each of the two RFUs@50MHz in the second extended TriMedia instance. That is, a lower pipeline frequency at the expenses of a double size FPGA is the trade-off of the second instance.

To perform color space conversion for an image, calls to CSC are issued within a software loop. The scheduled code when the RFU@100MHz is considered is presented in Figure 4. First, `LOAD` operations are issued to fetch the pixels in $Y'CbCr$ format from memory. Then, pairs of `CSC_R` + `CSC_GB` operations are launched to perform color space conversion, four pixels per call. After eight pixels have been converted, `PACK` operations reorganize the $R'G'B'$ information in 8-bit unsigned integer vectors. Finally, `STORE` operations send the results to a display FIFO.

According to Figure 4, 16 pixels can be processed with the latency 25 cycles. In order to keep the pipeline full, back-to-back CSC_R operation is needed. That is, a new CSC_R instruction has to be issued every two cycles (or, every four cycles in the RFU@50MHz–based instance). In this way, color space conversion can be performed with a throughput of 16/8 = 2 pixels/cycle. Unfortunately, this figure corresponds to the ideal case of infinite loop unrolling, which can never be achieved in practice. For a finite loop unrolling, the overhead associated to firing-up and flushing the CSC pipeline has to be taken into consideration. As a rule of thumb, the throughput drops to $N/(\textit{latency}/16 + (N-1)/\textit{ideal throughput})$, where N is the number of times which the loop is unrolled. For example, the ideal throughput drops to 1.3 pixels/cycle for $4\times$ loop unrolling, and to 0.64 pixels/cycle for a loop which is fully rolled. The same judgement can be carried out for the second RFU@50MHz–based instance. Since the results are pretty much the same, we will not go into details.

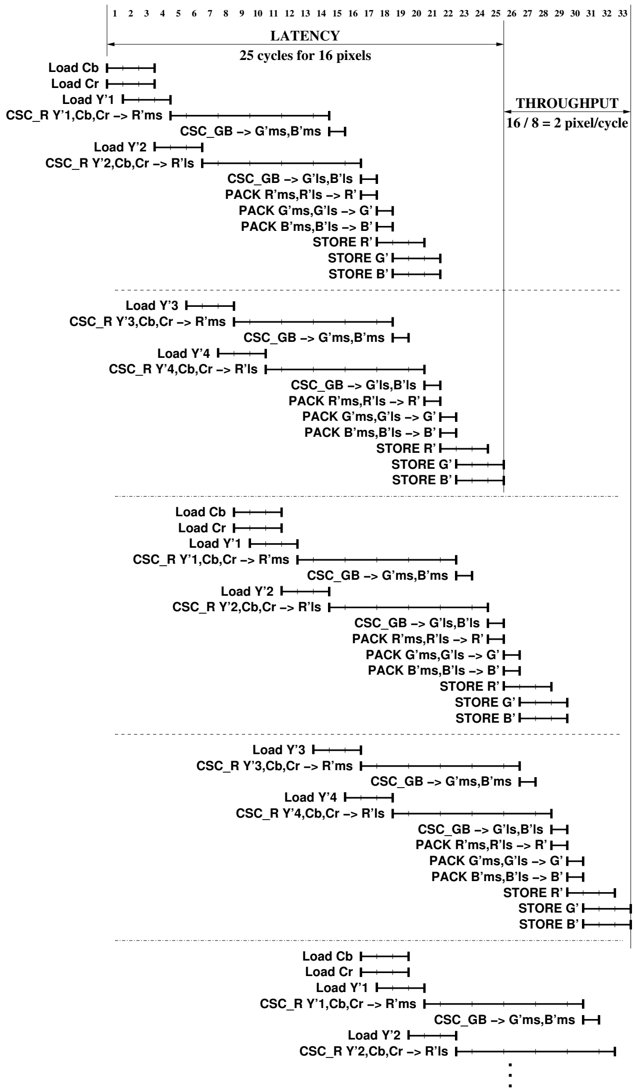

Figure 4. The scheduling result for a CSC unit having the latency of 10 and recovery of 2

The testing database for both pure-software and FPGA-based color space converters consists of a stream of two 256×256 pixel images, for which the Y', Cb, Cr components are stored in separate tables in the main memory. The data is organized as 16-bit signed integer vectors as resulted from motion compensation. The $Y'CbCr$-to-$R'G'B'$ conversion is done in an SIMD fashion, by sequentially processing four triplets of Y', Cb, Cr values at a time. As mentioned, *chroma* 2-fold upsampling is performed by means of a zero-order hold; this way, no additional processing is needed. Then, the linear mapping defined by Equation set 1 is carried out, and the result is sent to a display FIFO.

Since all the input data is stored into memory, the very first access to it (which is actually the single access in our application) will generate so called *compulsory* read cache misses. The sum of the data cache stalls and the instruction cycles needed to perform the proper color space conversion translates to a *worst case* scenario, which provides the lower bound of the performance improvements. We will also present the results according to the *best case* scenario, in which all cache misses are counted as part of the motion compensation process. That is, improvements in terms of instruction cycles required to perform strictly color space conversion will be reported.

The reference for evaluating the performance of the color space conversion carried out on FPGA-augmented TriMedia is a pure-software implementation on standard TriMedia [9]. The reference color space converter is implemented as a loop, where each iteration processes 16 pixels. Since

Table 1. Performance figures for $Y'CbCr$-to-$R'G'B'$ conversion

Experiment		1 RFU@100MHz		2 RFUs@50MHz	
	pure SW rolled	FPGA-CSC4 rolled	FPGA-CSC4 unrolled 4×	FPGA-CSC4 rolled	FPGA-CSC4 unrolled 4×
Instruction cycles	190,771	190,240	107,821	192,972	109,358
Instruction cycles / pixel	1.46	1.45	0.82	1.47	0.83
Pixels / instruction cycle	0.66	0.69	1.22	0.68	1.20
Instruction-cache stalls	937	697	1,342	702	1,368
Read data-cache stalls	90,112	90,112	90,112	90,112	90,112
Issues / cycle	4.81	3.47	3.84	3.35	3.95
I cycles + read D$ stalls	280,883	280,352	197,933	283,084	199,470
Instruction cycles / pixel	2.14	2.14	1.51	2.16	1.52
Pixels / instruction cycle	0.47	0.47	0.66	0.46	0.66

this pure-software implementation is beyond the paper scope, we will not go into further details. However, we still mention that by running our pure-software color space converter on a TriMedia cycle accurate simulator, we determined that an iteration which processes 16 pixels can be scheduled into 24 cycles, which translates into 0.66 pixels/cycle. It is also worth mentioning that 4.8 of 5 issue slots are filled in with operations in the pure-software implementation. This result is indeed a challenging reference for the TriMedia+FPGA hybrid.

Therefore, our experiment includes two approaches: *pure software* and *FPGA-based*. As mentioned, 0.66 pixels/cycle are decoded in the pure software approach, while 2 pixels/cycle can be decoded in the FPGA-based approach if the loop is unrolled an infinite number of times. The configuration of the RFU is carried out at application load time.

The $Y'CbCr$-to-$R'G'B'$ performance evaluation has been carried out considering two FPGA-augmented TriMedia instances: TriMedia + 1 RFU@100 MHz and TriMedia + 2 RFUs@50 MHz. A program has been written in C, and further compiled and scheduled with TriMedia development tools. To overcome the penalty associated to firing-up and flushing the pipeline, two techniques can be employed: (1) *loop unrolling*, and (2) *software pipelining*. Since the TriMedia scheduler uses the decision tree as a scheduling unit [3], all operations return their results in the same decision tree that they are issued, 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. Hence, only *loop unrolling* technique is considered in the sequel.

The loop calling the CSC instruction has been manually unrolled different numbers of times. The best results which corresponds to $4\times$ unrolling are presented in Table 1. We would like to mention that a $2\times$ unrolling does not suffice, since the firing-up and flushing overhead is still large. At the same time, an $8\times$ unrolling generates long decision trees, which in turn translates into reduced performance due to register spilling. As it can be easily observed, about the same figures are obtained for both FPGA-augmented TriMedia instances. The speed-up is $\frac{0.66-0.47}{0.47} \times 100 \approx 40\%$ according to the *worst-case* scenario, and $\frac{1.22-0.66}{0.66} \times 100 \approx 85\%$ according to the *best-case* scenario.

Finally, we would like to mention that a more realistic *average-case* scenario will assume a memory access pattern for reducing the number of read cache misses. However, this is subject to optimization at a complete MPEG decoder level, and therefore, beyond the paper scope. Thus, we do not have statistically relevand data for the time being. Since we are not able to make a reliable estimation according to the *average-case* scenario, we proceed to a conservative evaluation by taking into account the entire number of read cache misses (worst-case scenario), and claim that the FPGA-augmented TriMedia/CPU64 can perform color space conversion 40% faster than the standard TriMedia/CPU64. 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 [14], such an improvement within the target media processing domain indicates that the TriMedia + FPGA hybrid is a promising approach with respect to color space conversion.

6. Conclusions and future work

We have described an $Y'CbCr - to - R'G'B'$ converter on FPGA-augmented TriMedia. For such a task, the lower bound of the performance improvement over the standard TriMedia is 40% 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. In future work, we intend to consider performant anti-image filtering in the process of chroma upsampling.

Acknowledgements

This project is supported by the doctoral fellowship RWC-061-PS-99047-ps from Philips Research Laboratories in Eindhoven, The Netherlands.

References

[1] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG-2*. Kluwer Academic Publishers, Norwell, Massachusetts, 1996.

[2] J. R. Hauser and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 12–21, Napa Valley, California, April 1997. IEEE Computer Society Press.

[3] J. Hoogerbrugge and L. Augusteijn. Instruction Scheduling for TriMedia. *Journal of Instruction-Level Parallelism*, 1(1), February 1999.

[4] International Telecommunication Unit. Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios. ITU-R Recommendation BT.601-5, October 1995.

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

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

[7] C. Poynton. *A Technical Introduction to Digital Video*. John Wiley & Sons, January 1996.

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

[9] M. Sima. Color Space Conversion on VLIW platforms. Private Communication, August 2002.

[10] M. Sima, S. Cotofana, J. T. van Eijndhoven, S. Vassiliadis, and K. Vissers. 8×8 IDCT Implementation on an FPGA-augmented TriMedia. In K. L. Pocek and J. M. Arnold, editors, *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, Rohnert Park, California, April 2001. IEEE Computer Society Press.

[11] M. Sima, S. Cotofana, S. Vassiliadis, J. T. van Eijndhoven, and K. Vissers. MPEG Macroblock Parsing and Pel Reconstruction on an FPGA-augmented TriMedia Processor. In A. Jacobs, editor, *IEEE International Conference on Computer Design*, pages 425–430, Austin, Texas, September 2001. IEEE Computer Society Press.

[12] M. Sima, S. D. Cotofana, S. Vassiliadis, J. T. van Eijndhoven, and K. A. Vissers. MPEG-compliant Entropy Decoding on FPGA-augmented TriMedia/CPU64. In J. Arnold and K. L. Pocek, editors, *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2002)*, pages 261–270, Napa Valley, California, April 2002. IEEE Computer Society Press.

[13] P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[14] 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. E. Vranken. TriMedia CPU64 Architecture. In *Proceedings of International Conference on Computer Design*, pages 586–592, Austin, Texas, October 1999. IEEE Computer Society.

[15] R. D. Wittig and P. Chow. OneChip: An FPGA Processor With Reconfigurable Logic. In K. L. Pocek and J. M. Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 126–135, Napa Valley, California, April 1996. IEEE Computer Society Press.